PEGA ACADEMY™

PRPC: Fast Track Lesson Companion

PRPC Version:6.2 SP2

Course ID: FT62SP2V1

This document is the property of:

Pegasystems Inc.

101 Main Street

Cambridge, MA 02142-1590

Phone: (617) 374-9600

Fax: (617) 374-9620

www.pega.com

Document: PRPC: Fast Track Lesson Companion

Course ID: FT62SP2V1

Updated: 5/9/2012

Table of Contents

Introduction to Fast Track

Audience

This course is for System Architects who:

- Want to learn how to create multi-process applications that will allow users to manage complex business processes
- Are interested in building applications for reusability

Prerequisites:

These prerequisites are recommended but not required.

- Application development experience
- Knowledge of your company's business practices and policies
- Familiarity with one or more project methodologies

Course Objectives

At the end of this course you should be able to:

- Describe the components of a BPM application
- Explain the benefits of using PRPC to create an application
- Modify an existing application by defining properties, modifying the user interface, and creating business rules and processes

How to Use this

This guide contains the following sections:

- **Modules** — Are a large grouping of related lessons.
- **Lessons** — Are the things you need to know about a specific topic.

Lessons:

Each lesson contains information to help you understand the benefits and features of PRPC. Lessons always contain an Objective and Things to Know section. The Things to Know section complements the information presented by your instructor and provides additional detail, including references to the Pega Developer Network (PDN) or Developer Help for additional guidance on features or functionality.

User ID and Password

To complete the exercises, you often play different roles. To do this, you log onto the system as a different operator so that you can perform the tasks associated with the role. Below is a list of roles, User IDs, and passwords.

Role	User ID	Password
System Architect	Architect	password
System Architect	SystemArchitect@ILend.com	rules
End User	User@iLend.com	password
End User	JuniorLoanOffi cer@iLend.com	rules
Manager	LoanOfficer@iLend.com	rules

Module 1: Introduction to BPM and PRPC

Lessons Covered:

- BPM Overview
- PRPC Overview
- Operators, Roles, and Portals
- Creating and Processing Work
- Retrieving Work
- Organizational Structure
- Designer Studio

BPM Overview

Objectives

At the end of this lesson, you will be able to:

- Define Business Process Management
- Define the terms work, work type, and work item
- Describe the elements of a BPM application

Things to Know

Business Process Management (BPM)

A **business process** describes and controls how business is conducted. It defines how the data and information flows and what interactions occur both internally and externally. **Business Process Management** includes the methodologies, technologies and tasks that connect all the information, interactions and assets involved in executing a business process. It includes:

- People, documents, workflows, and applications
- How work gets routed through a business process to its completion according to defined business rules
- Planning, designing, building, operating, maintaining, and improving the business process

As work is performed, data is collected, processed and presented to users and permanently stored. The **data model** defines the required information and describes the relationships between the data used for interactions within the organization as well as external exchanges across the extended supply chain. It is the foundation for any BPM application.

The categories which help us organize different elements or components of a Business Process Management application include:

- **Process** — represents a workflow and is comprised of business rules that determine how work is completed and resolved.
- **User Interface** — represents the design, display and management of information on user portals, forms used to collect information, pages used to display information, and any other presentation media used by the application.
- **Logic and Decisions** — provides us with the ability to evaluate information, perform calculations, route work, and automate decisions/logic to enforce the business rules.
- **Integration** — provides us with the tools to share data with other internal or external systems and to connect to and work with multiple systems of record.
- **Reporting** — enables us to monitor work status and track progress through the use of predefined reports and tools to create new personalized reports.
- **Case Management** — provides the tools that enables us to consolidate and coordinate the processing of related work items as well as define the structured and unstructured content associated with the cases managed by the application.

Work

Work represents a process-driven business transaction that you want to manage and is something that may require information to be entered, decisions to be made, calculations to be completed, or other tasks to be performed for the work to be resolved or achieved. For example, lets assume we want to open a new account at our favorite store — Amazon.com. **Opening an account is work.** In order to complete the opening of an account, we must do things to the

work which may require getting help from other people or systems. Below is a list of things that we might want to do to the work to complete (or resolve) the opening of the account. We may:

- Enter information about the person who wants to open the account
- Assign the work to another person to verify
- View the information entered
- Search for information
- Approve the account opening
- Process the request to create an account
- Reopen the account record for review
- Resolve or complete the task

Work Type

A **work type** is the **fundamental unit of work** to be processed and resolved. In our example, the new account request is our work type. The work type provides us with a description of the work. Examples of work types include:

- Address Change Form
- Insurance Claim
- Account Open / Member Enrollment Form
- Customer Support Incident Report

Work Items

Once we know what our work types are we can create work items. **Work items are a unique instance of work that we want to process and resolve.** When we use an application we create, update, and eventually resolve work items. Every work item in PRPC has a unique ID, an urgency value, and a status.

PRPC Overview

Objectives

At the end of this lesson, you should be able to:

- Explain PRPC's 6 R's
- Discuss process-driven PRPC applications
- Locate and be able to use the Pega Developer Network

Things to Know

PRPC 6 R's

PRPC applications provide process management and automation through six functional capabilities, informally known as the Six R's:

- **Receiving** — Accept and capture the essential data that multiple sources describe as work
- **Routing** — Use characteristics of the work, together with knowledge about the workforce, to make intelligent matches and assignments
- **Reporting** — Provide real-time visibility into work progress, productivity, bottlenecks, and quality
- **Responding** — Communicate status, requests for information, and progress to the work originator and to other interested people involved in the work
- **Researching** — Support analysis and decision-making by providing access to external systems and databases
- **Resolving** — Complete the work, then update downstream systems promptly through automated processing and automated support of users

Process-Driven PRPC Applications

In PRPC, applications are process-driven; a work item is guided through its lifecycle by a process, obtaining the information needed to make decisions, then routing the work to others who help complete the work. To accomplish this, we capture processes as diagrams that PRPC can execute. Using the process diagrams, we can see, understand, and change the behavior of the application directly.

Pega Developer Network (PDN)

The PDN is a member-restricted resource available only to customers, partners, and Pega employees. It provides useful information about installing, developing, and supporting your Pegasystems solutions. Features include announcements about upcoming events; forums, communities, and FAQs; Online Help, Knowledge Base articles; hotfixes and service requests; University of Pega training information; upcoming and archived webinars; and Pegasystems product documentation.

To request a new PDN Account, go to http://pdn.pega.com. Use your **work e-mail account**, rather than a Yahoo, Gmail, etc. account. Requests for new accounts are processed within two business days.

Some features of the PDN include:

- **Browsing** — use the Browse menu to drill down into topics.
- **Searching** — use the Search tool to find articles. Click a Search Suggestion to open the corresponding Knowledge Base (KB) article. Click the magnifying glass to see a list of all the related material on the PDN.
- **Forums** — discuss technical issues with other PDN members.

- **eLearning** — self-study courses are available on the PDN through the University of Pega menu.
- **Support** — enter support requests or download hotfixes for known issues.

Operators, Roles and Portals

Objectives

At the end of this lesson, you will be able to:

- Define an operator
- Define a project role
- Define a portal
- List Pega's standard project roles
- List Pega's standard portals
- Describe the relationship between operators, project roles, and portals

Things to Know

Operators

Anyone that uses PRPC is an operator, sometimes called a user. Each operator has a unique operator ID, sometimes called a username, that they use to log into PRPC. Operator IDs are often email addresses or the same username used by an LDAP directory.

Roles

Roles define implementation team and end user responsibilities. There are four primary PRPC project roles.

1. **Work users**, also known as end users, are the primary users of the PRPC applications we build. They create and resolve work. Claims adjusters and customer service representatives are work users.
2. **Work managers** manage the work users, and typically do everything the work users do plus monitor activity and reporting data through our applications. Call center managers are work managers.
3. **Business architects** are responsible for capturing business objectives, defining business processes, creating and managing business rules. They are sometimes called business analysts or system analysts.
4. **System architects** are responsible for implementing our systems based on the requirements defined by the business architects. They are sometimes called developers.

Other roles include project managers, system administrators, executive sponsors, and QA/UAT testers.

PRPC Portals

PRPC portals provide a role-based user interface to allow operators to do the things they need to do. There are four standard PRPC portals:

1. The **Case Worker** portal allows work users to enter, update, and resolve work items and cases.
2. The **Case Manager** portal allows managers to view the system-maintained list of work assignments.
3. The **Business Analyst** portal allows business architects and business analysts to define the high-level processes that must be resolved.
4. **Designer Studio** allows system architects to build the application's data, logic, UI and process rules.

Portals are defined by a collection of interacting user interface rules, and are browser-based. They display different physical layouts and content to different users.

The standard portals can be customized to reflect the terminology, styles, layout, and facilities appropriate to each project role, and completely custom portals can be constructed from scratch.

The Relationship Between Operators, Project Roles, and Portals

Each operator can have one or more project roles. For example:

- Cathy is one of three system architects on an implementation team.
- Hani is both a business architect on the implementation team and a work manager.

Each operator can have access to one or more portals:

- Because Cathy needs to build the application and view it as an end user, she has access to Designer Studio, the Case Manager portal, and the Case Worker portal.
- Hani has access to both the Business Analyst portal and the Case Manager portal.

Creating and Processing Work

Objectives

At the end of this lesson, you will be able to:

- Create a work item
- Process a work item
- Resolve a work item
- Use the Where Am I? button
- Review the audit trail

Things to Know

Creating a Work Item

In order to complete a business transaction, such as opening an account or filing a customer service request, it may need to be reviewed by people within the organization and logged. In PRPC, application users create, update, and eventually resolve work items. A work item, sometimes referred to as a work object or case, is the primary unit of work that needs to be completed in an application, and the primary collection of data on which the flow operates.

System architects design work types that define the information required for each transaction, such as name, date of birth (DOB), address, and social security number (SSN), and define the steps in the process. To create an active instance of the work type – called a work item – users, such as customers or customer service agents, must fill out the work item's form fields with the pertinent information and submit it. The flow (business process) creates the work item.

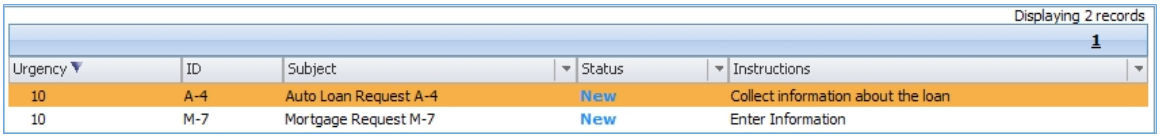
Processing a Work Item

Submission of the work item form starts the processing of that work item. As a work item advances through a flow via user input and automated processing, more information gets added to the work item's properties. The processing of the work item includes assigning tasks to people within the organization, handling data retrieval and updates, and executing automated decisions. The flow controls the process of creating the work item and drives the work item from assignment to assignment and through the business process to resolution. An assignment may present the user with a mandatory form or with a choice of actions, each with its own user interface. Each of the possible user actions is called a flow action. Users can also include notes and attachments throughout the lifecycle of a work item.

For every work item, PRPC defines certain information, or properties, such as a unique identifier, a default urgency, and a status. As the work item is driven through the process toward resolution, PRPC updates its status. The possible values of a work item’s status, such as new, open, pending and resolved, are restricted and controlled and can only be changed through specific mechanisms.

Worklists

A worklist is a list of open, outstanding assignments ready and waiting for a specific user to perform them. As a PRPC process executes, assignments are created for the work items. These assignments are then routed to users, who complete the assignments to advance the work item through its process. When a user first logs in, PRPC displays the user's worklist (from the User portal), as shown in the example below.



The worklist usually includes a column that identifies a work item ID, but it's important to remember that a worklist is not a list of work items, but rather, a list of assignments that are pending for the user. Each user has their own worklist, maintained independently by PRPC and updated automatically as work items are processed.

A user can click on one of their assignments to open the work item and perform the assignment. When a user submits a form that completes the assignment, PRPC advances the work item through its process to the next step. If this next step is another assignment, that assignment is added to the assigned user's worklist, and the process repeats.

Assignments on a worklist are displayed in order of urgency, with the most urgent assignment appearing first on the worklist.

One work item may have, at any one moment, none, one, or multiple open assignments. Multiple assignments may all appear on one user's worklist, or may be distributed among the worklists of various users, or in one or more workbaskets.

Workbaskets

A workbasket is a named queue of open assignments that are not currently associated with an operator. Think of a workbasket as a common bucket, available to all members of the work group. These assignments are awaiting processing and ordered in decreasing urgency, similar to the contents of a worklist. These assignments are sometimes known as the work queue.

Users generally gain access to a workbasket through their assigned work group. Users with access to a specific workbasket can open assignments from that workbasket and complete them, just as they do with assignments on their own worklist.

Removing Work from a Workbasket

Assignments are actively moved from the workbasket to a user's worklist in one of three ways:

1. Qualified users can remove an assignment from the workbasket to process the assignment. When this happens, the assignment is removed from the workbasket and added to the user's worklist.
2. Applications can automatically route (move) the assignments in a workbasket to users' worklists based on work schedule, due date, skills, workload and other factors.
3. Managers can transfer an assignment from a workbasket to a user's worklist, reflecting the manager's decisions and judgement about who best should work on it.

Resolving a Work Item

After the last assignment and all work on a work item has been completed, its status is set to Resolved. The work item and all of its associated information, including how it was processed, is stored in PRPC's database. After a work item is resolved, it generally cannot be modified until it is reopened, although some changes are possible.

Where Am I?

While working on the item, you can click the Where-Am-I? icon on the form to view a diagram of the current flow rule in a separate window and see where the work item is currently in the business process.

Viewing the Audit Trail

The audit trail allows the end user to view the history of a work item, including information such as:

- Which operators have worked on the work item and when
- What automatic processes were performed by the application

By default, PRPC automatically adds an entry to the work item history at each step in the process. We can also add notes to the history to provide additional information about a particular step.

Organizational Structure

Objectives

At the end of this lesson, you should be able to:

- Describe an operator's organization, division, and unit
- Describe an operator's work group
- Explain how work groups can facilitate work processing

Things to Know

Every employee is part of their company's **organizational structure** (commonly referred to as an *org structure*). The organizational structure usually represents a division of labor along functional lines. PRPC implements a three-level organizational structure composed of the following levels:

- **Organization** — This is the highest level in the organizational structure, and encompasses the entire organization or corporation. At this level, you can specify a company-wide calendar (to manage work days and days off) and a Chief Financial Officer (CFO) or Controller for accounting purposes.
- **Division** — This is the middle level in the organizational structure. A division is the most general functional level within an organization and can be associated with cost/profit centers within an organization.
- **Organizational Unit** — The organizational unit is lowest level in the organizational structure. An organizational unit (or *org unit*) represents a more specific functional level than a division. An organizational unit can be defined as part of another organizational unit, allowing us to define as many organizational levels as needed to map out an organizational structure.

PRPC can leverage a company's organizational structure, for correspondence and for sharing rules. Every operator ID identifies that operator's organizational structure, which means that an operator's position within a company's hierarchy is always known.

Related to the organizational structure — though not part of it — is the **work group**. The work group describes an operator's **capacity to receive assignments and process work**. Each work group is assigned a manager, to whom assignments can be escalated automatically when needed. Work groups are also associated with workbaskets. Our application can use this association to automatically populate an operator's worklist with assignments transferred from the workbasket that is associated with their work group.

PRPC displays the organizational structure and work group for an operator on their **operator profile**.

Designer Studio

Objectives

At the end of this lesson, you will be able to:

- Navigate the Designer Studio
- Use the What's Happening event feed
- Use the Pega button to locate and use gadgets
- Locate rules using the Application Explorer

Things to Know

What is Designer Studio?

Designer Studio is a development environment for building applications that provide intuitive navigation of application rules and development tools. It also brings visibility and structure to existing elements that we define in an application.

Navigating Designer Studio

Application Menu

The Application menu shows the name of the current application and lets us:

- Switch applications and access another application
- Access the Application Profiler and Application Accelerator to create new applications or extend an existing application
- Switch work pools

Profile Menu

We use the Profile menu to view our operator profile and update our preferences. From the Operator menu, we can also access our operator and access group records, the application rule for the active application, favorite rules that are delegated to us, and shortcuts to activities and certain types of reports.

Tabbed Rule Navigation

Each form is represented by a tab. Using this feature, we can navigate to home, rules, wizards, lists, and work. The default maximum number of open tabs is 20, but we can change this value through Desktop Preferences. Scrolling over each tab displays type, rule, and RuleSet Version information for tabs that represent a rule.

Quick Launch Toolbar

The Quick Launch toolbar provides quick access to frequently used functionality such as the Run menu, Clipboard, Tracer, Help, and My Checked Out links.

Developer Help

Developer Help provides in-depth reference documentation to guide us in building a PRPC application. Access it via the Help menu.

Context-Sensitive Help

Context-specific help provides topic-specific guidance on how to create and use rules. We can access this kind of help from individual rule forms and landing pages.

Global Search

We use the Global Search feature to find rules, data instances, operator IDs, access groups, Pega button menu commands (such as the Branding wizard), and Help topics.

What's Happening Event Feed

The What's Happening event feed provides near-real-time display of designer activities and progress to improve collaboration. This feature displays the following events: rule actions (check out, check in, delete, delete check out) and comments. Use What's Happening to post messages and questions. This Display is limited to the last 25 messages. To see the most recent messages, refresh the home page. We can also filter the display by event type; for example to show only requirements and use cases, flow rule, and application designer posts.

The Pega Button

The Pega button provides quick access to development tools, called gadgets. Each gadget provides system or application information and is grouped with similar gadgets into a landing page. Landing pages bring visibility to helpful information about our application and make that information easily accessible. While built-in landing pages are not customizable, we may create our own landing pages to show specific groups of gadgets. Many gadgets allow configuration of specific capabilities in our system or our application.

The Navigation Panel

The Navigation panel organizes rule instances by:

- Application and work type (Application Explorer)
- Checked out rules, personal favorites, and access group favorites (My Rules)
- Concrete Applies To class (Class Explorer)
- Rule type (Rules Explorer)
- Work with the application from a business assets perspective using the Profile Explorer

Application Explorer

In the Application Explorer, rules are grouped by work type (also known as a class or class instance). We use the Class Selector to switch between application classes. Commonly selected classes are grouped as "Best Bets".

Profile Explorer

The Profile Explorer gives Business Architects and others responsible for delivering the business benefits of the application quick access to business-related assets and operator profile-related elements. With its metric-related display, we can monitor the application's development to ensure its design and behavior meet the specifications and requirements and assess its progress over time.

Module 2: Building a Basic Application

Lessons Covered:

- Introduction to Direct Capture of Objectives
- Introduction to Data Elements
- Introduction to Business Rules
- Introduction to Process
- Introduction to User Interface

Introduction to Direct Capture of Objectives

Objectives

At the end of this lesson, you will be able to:

- Define the three primary Direct Capture of Objectives (DCO) tools
- Explain how the DCO tools and processes fit into common development methodologies
- Use a Process Discovery Map to create an initial application

Things to Know

When building an application, the most important step is to capture the process as accurately as possible. There are several tools that we can use to do this, but the concept is the same no matter which tool we use.

Direct Capture of Objectives

The process of capturing the information that will be used to build an application is called Direct Capture of Objectives. **DCO is** a process of gathering information about the business process, allowing the implementation team — IT, testers, and business stakeholders — to collaboratively build the application; it **is not** a methodology, a tool, or a discreet event.

When we use DCO, we find ourselves using new terms, and using familiar terms in a new context. The terms we most frequently encounter during the DCO process are described below.

Business Objective

A business objective represents the response to the demands placed on a business by internal and external forces. Business objectives are the goals of the project that, when reached, provide a return, or business benefit.

Requirement

A requirement is a detailed bit of information that can be used as a checkpoint, or guide, to ensure that the application behaves properly. For example, a loan request application might have a requirement that the income for an applicant must meet a specific threshold.

Actor

An actor is the entity that performs a task as part of the process. An actor can be a person, an external system or PRPC itself.

Atomic Use Case

Atomic use cases (commonly shortened to *use cases*) represent information about a specific task. They should be small and granular. Ideally, atomic use cases are defined for a single work type and executed by a single actor. For example, a rejection is performed by one user (though the user performing the rejection may vary depending on circumstances). Use cases correspond to various PRPC rules, which are built out during the development process.

Work Type and Support Type

A work type is a template of the work that is to be performed. Work types are instantiated into work items, which are then processed and resolved. Work types are associated with one or more atomic use cases.

A support type is similar to a work type, except that it does not instantiate into a work item. Support types allow you to collect use cases that represent ancillary functionality. You can also use the Common supporting type to collect use cases that are shared between work types.

DCO Tools

As part of the DCO process, development teams use the following tools. Not everyone on the team will use all of these tools; in fact, some members of the development team may not use **any** of these tools. But, even if they do not use any of these tools, the work they do may impact or be affected.

Application Profiler

Business architects and business analysts use the Application Profiler to capture high-level information about the application, such as work types, atomic use cases and interfaces. The Application Profiler creates a special work item called an Application Profile (commonly referred to as an AP). This AP can be used by the Application Accelerator to generate the starting point of the application. In addition, we can also generate a print version of the AP, called an Application Profile document, which we can circulate to interested parties who may not have access to PRPC.

Application Accelerator

The Application Accelerator (AA) automates the creation of an application foundation — the starting point from which you proceed to develop an application. Typically used by Senior or Lead System Architects, the Application Accelerator captures details about the implementation used to create the basic elements of your application, such as a class structure, draft flows, and correspondence rules. Additionally, we can use a completed AP as a starting point for our Application Accelerator to incorporate the information (such as work types, atomic use cases, and requirements) collected by the Application Profiler.

Application Documentation Wizard

We use the Application Documentation wizard to document the rules that make up the application. This ensures that future application architects will be able to understand how the application — and its component rules — work, making it easier to perform maintenance and upgrade the application. The wizard captures the information on the History tab of a rule, so it is important that we complete the Full Description and Usage fields for each rule.

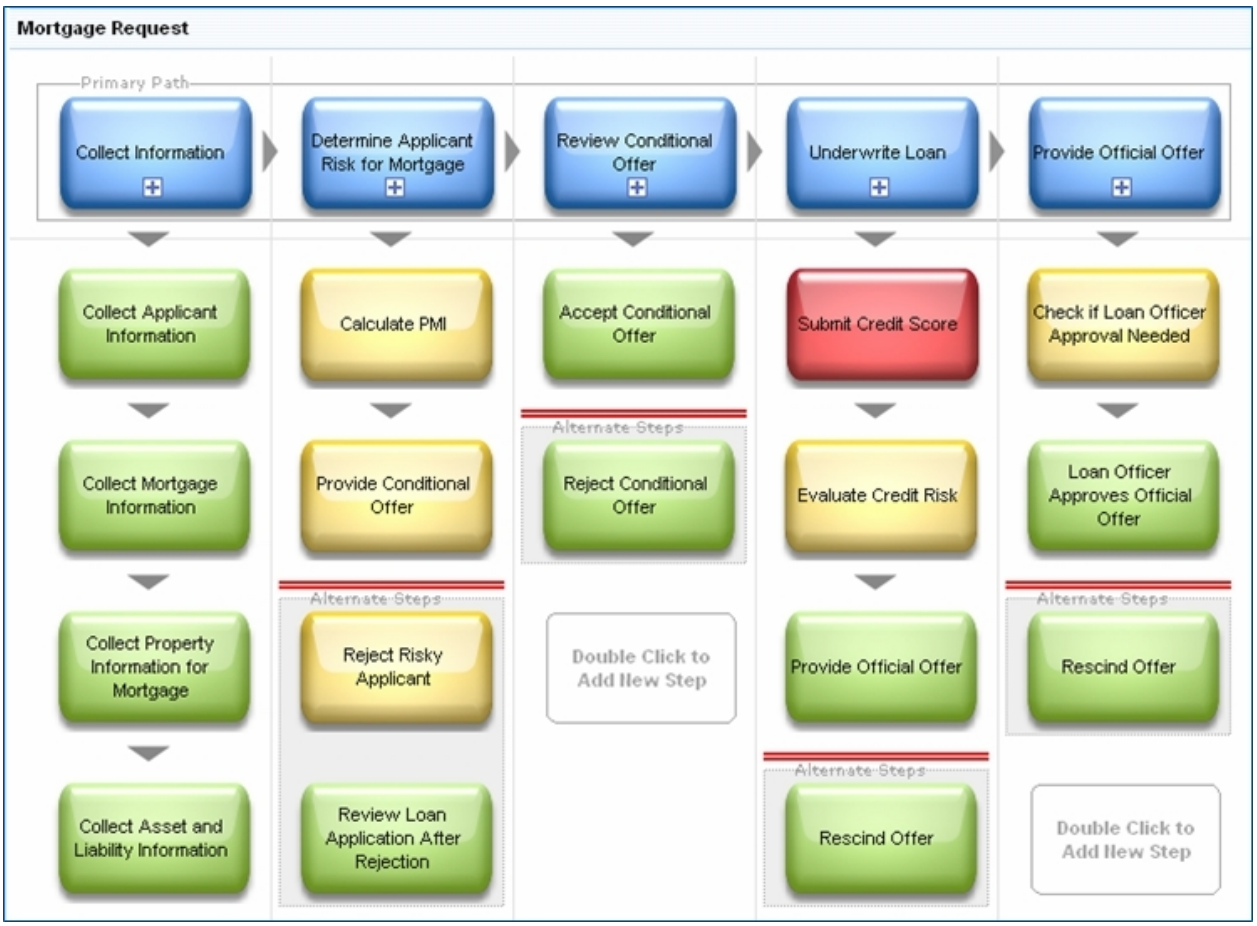
Application Express

We use Application Express to quickly prototype our application, without getting bogged down in the details of use cases and requirements. It allows the development team to quickly "mock up" what the application should look like, and provides a reference point for members of the implementation team when they begin to scope out the details of the application.

Once this prototype is completed, the implementation team uses the Application Profiler to capture the use cases and other information for the application, and then the Application Accelerator to build the application skeleton (which the team will "flesh out" during the development phase of the project). Over the next few lessons, however, we'll use Application Express to quickly build an application, to help us understand the process of application development using PRPC.

Process Discovery Maps

The Process Discovery Map captures the manual and/or automated steps in a process. It provides an at-a-glance view of all use cases that make up a process. These steps are organized in primary and alternate paths, and and can be grouped into sub-processes to reduce complexity and allow for reuse. Use cases are color-coded so stakeholders can quickly see how they are performed.



A Process Discovery Map is not a **replacement** for flow modeling, which encompasses decisions, looping, routing or other details that come out of subsequent process refinement sessions. Rather, it allows stakeholders to capture the main steps in a process, so the implementation team can get to a sufficient level of detail necessary for project sizing, without over-analyzing the entire solution.

When creating a process, always remember to think in terms of the primary path — the sequence of steps that users will most likely perform to resolve a work item. The primary path is the **top** row of the Process Discovery Map.

To add a step to the primary path, double-click the appropriate placeholder to add a shape. Then:

1. Specify the use case.
2. Select the shape type (human-based step, automated step, integration, subprocess).
3. Provide details about the use case and list any associated requirement.

To create an alternate path, right-click the preceding shape and select Add New Alternate Step.

When we add a step below a shape in the primary path, the primary path shape automatically becomes a subprocess (if it isn't already), and the new shape is added to the subprocess as part of its primary path. For simple processes, we may not need any subprocesses in the primary path. However, with a more complex process, collecting use cases in smaller, reusable subprocesses makes the Process Discovery Map easier to read and understand.

Building the Application

When we finish the Process Discovery Map, we can create our application. The AP and AA involve additional steps to create an enterprise application, but the information collected in those steps is either optional or is automatically determined from the information provided as part of the Process Discovery Map. Note that information provided by the Discovery Map can be overridden.

The resulting application is functional; we can run the process from start to finish. We'll need to provide additional elements, such as UI fields and process steps, but the **process** can be completed. This allows us to verify that what we *are* building is what we actually *want* to build.

We can update the application as we develop it. If additional use cases and/or requirements become necessary, we can manually add them to the application. This allows us to continue the direct capture of process, UI, and business rule requirements as needed.

Introduction to Data Elements

Objectives

At the end of this lesson, you will be able to:

- Define data
- Define a property
- List two ways to create properties
- Define a field
- Describe the relationship between properties and fields
- Define a data model

Things to Know

There are four key components of a BPM application.

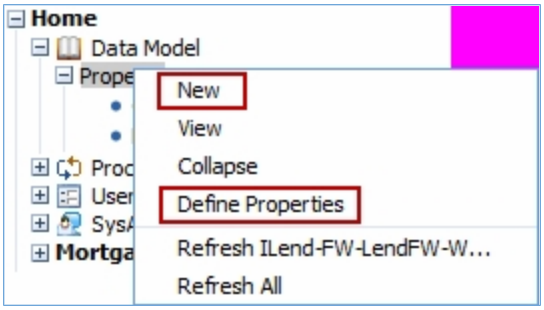
Data is the information in our application. The acronym **CRUD** defines how we typically work with data in our application:

- We **create** data by gathering it from end users in user interface form fields.
- We **retrieve** data from databases and display it in our user interface either in a read-only or in a format that allows for updating.
- We **update** data by sending modified form field data back to the database.
- We **delete** data by selecting one or more records in our user interface and removing the corresponding records from a database.

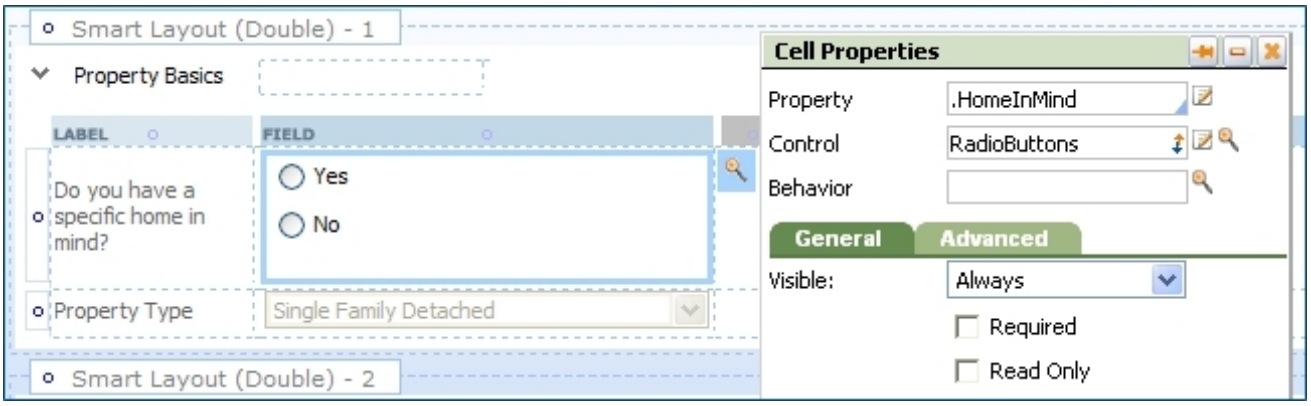
Properties are the containers that hold our application data. Each property must have a name and may or may not have a value as shown in the table below.

Property Name	Property Value
FirstName	Ben
LastName	Navarrao
City	Portland
State	(null)

We **create** properties in PRPC by creating **property rules**. Property rules are found in the **Data Model** rule category. We can create properties one at a time using the New Rule dialog, or in bulk using the **Define Properties Wizard**.



Fields (or form fields) are controls used to collect new data, display existing data for updating, or display existing data for deletion. Fields reference properties. Here, a radio button field references the HomeInMind property.



A **data model** defines what properties our application needs and where they need to be defined. If our data model is known, we can create our properties first and then reference them in our fields. However, we often mock up our UI with fields first and then define and link them to our properties later.

Introduction to Business Rules

Objectives

At the end of this lesson, you should be able to:

- Define a business rule
- List four business-friendly decision rules
- Compare and contrast decision rules
- Explain how a decision table is evaluated

Things to Know

Business rules are statements of how we do things (statements of policy). For example, we may have the following business rules:

- Unless prior authorization is obtained, brand name prescriptions are not covered when generic equivalents are available.
- All expenses over $5000 must be approved by a vice president.
- Time sheets submitted after Sunday at midnight Eastern Time are considered late.

Applying business rules in our PRPC applications can require an array of PRPC functionality, but there are four key logic-related rule types we commonly use. All four can be used in conjunction with the Decision shape in a flow diagram.

1. **Decision tables** use a spreadsheet format to look at 1-n conditions, defined by the same 1-n properties, to return a result.
2. **Decision trees** use a nested logic format to look at 1-n conditions, defined by different properties, to return a result.
3. **Map values** use a spreadsheet format to look at 1-n conditions, defined by the same 1 or 2 properties, to return a result.
4. **When rules** use a nested logic format to look at 1-n conditions, defined by different properties, to return true or false. Unlike other decision types, when rules are called from connectors. These connectors can only be used when the Decision shape is set to the Fork option.

These four rule types allow you to define and automatically apply the logic of your business rules. They present the logic typically defined in low-level code in a format that is easily delegated to a system or business architect. All four rule types are found in the Decision rule category in the Application Explorer, and are sometimes collectively referred to as decision rules.

Decision tables are discussed below. Decision trees and when rules are covered in more detail in their respective lessons. Map value rules are seldom used; decision tree rules provide a similar format with more flexibility.

Decision Tables

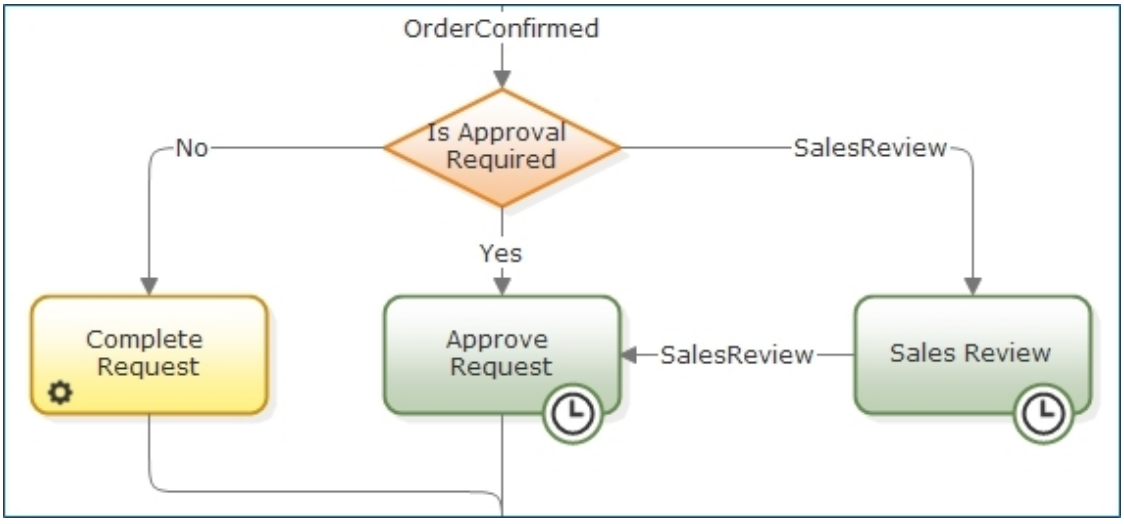
As defined above, decision tables *use a spreadsheet format to look at 1-n conditions, defined by the same 1-n properties, to return a result*. Let's deconstruct this definition with the following simple decision table that defines when a purchase request requires approval.

- **Use a spreadsheet format** — This decision table has four columns and five rows. It can be edited in Excel.
- **To look at 1-n conditions** — The above decision table's rows define four conditions, one for the if row, two for the else if rows, and one for the otherwise row.
- **Defined by the same 1-n properties** — Each row in the decision table above looks at the value of the Bill To Department property and the Unit Price property.
- **To return a result** — If a particular condition row is true, the value in the return column is returned — here either No or Yes — and processing stops.

Each condition is evaluated sequentially, from top to bottom, until one of the conditions evaluates to true, and a result is returned. Putting it all together, here is how the above decision table runs:

1. At some point in our application, we need to know if a purchase request requires approval, so we call the IsApprovalRequired decision table.
2. If the *Bill To Department* for the purchase request is Sales and the Order Total is anything, we return No — approval is not required. If that's not what's happening...
3. If the *Bill To Department* is anything and the Order Total is less than $100, we return No — approval is not required. If that's not what's happening...
4. If the *Bill To Department* is Admin and the Order Total is less than $200, we return No — approval is not required. If that's not what's happening...
5. We reach the otherwise condition and return Yes — approval is required.

Decision tables are most commonly called when a work item encounters a decision shape. Which path the work item takes as it leaves the decision shape depends on the result the decision table returns; the decision shape has one exit path for each possible return result. For example, the decision table associated with the following decision shape has three return values: No, Yes, and SalesReview.



Whenever possible, use decision tables for frequently changing logic that is delegated to business users. They find the Excel-like format of decision tables familiar and the logic easy to understand.

Introduction to Process

Objectives

At the end of this lesson, you should be able to:

- Describe how processes are defined in PRPC
- List the common flow shapes and when they are used
- List the key rules associated with the common flow shapes
- Add a new shape and its associated rule to an existing flow

Things to Know

What Is a Flow

A **flow** is a fundamental rule type that **implements all or part of an application's business process.** A flow, also called a process or workflow, provides a visual representation of the flow relationships among shapes and connector arrows. The flow diagram also shows relationships among a starter flow that creates a new work item and subflows that the starter flow may call or start.

The flow, its shapes, and connectors reference rules and provide parameters and values that control many aspects of the business process:

- How a work item is created
- The sequence of steps the work item follows as it advances through the business process that the flow implements
- Who can perform the work
- How the flow operates on the work item's properties
- What decisions and actions occur automatically, and
- How to resolve (complete) the work item.

When first created, each flow rule has only a single start shape, an Assignment shape, and an end shape. When fully developed, a flow may define many optional detours and branches, to allow different work items to follow different paths depending on the outcome of a decision and available resources. For example, a flow can route a work item to a different customer service agent or require an additional level of approval when a property value exceeds a specified threshold.

As a flow executes, it typically generates a unique ID for each work item. When data or decisions are needed, the flow creates assignments — tasks that must be performed by users, groups, agents, or external participants — and places the assignments in a worklist or workbasket. Required data is obtained from users or other sources, while some steps, such as calculations and decisions, are performed automatically to advance the work item through the flow until it is resolved or reaches an end shape. Where appropriate, an application may implement straight-through-processing, meaning a flow that runs from start to an end without requiring any user intervention.

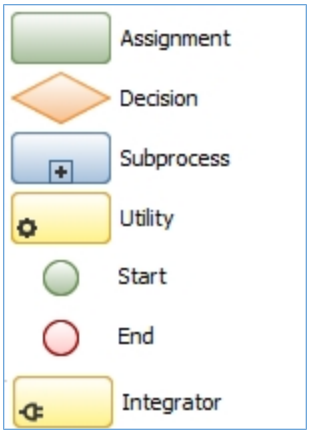
You can delegate selected flow rules to business managers to allow them to update and evolve their flow.

Process Modeler

PRPC's native flow editor, Process Modeler, uses notation based on the Object Management Group's Business Process Modeling Notation (BPMN) standards. Process Modeler allows graphical definition of flow elements and relationships. It is a thin-client editor with natural drag-and-drop operations that you can use to create, arrange, and update flow shapes and connectors.

Basic Flow Shapes

The following menu shows the basic Process Modeler shapes.

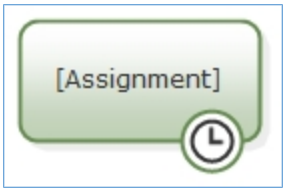


Start

The start shape begins every execution of the flow. Each flow can have only one start shape.

Assignment

Use the assignment shape to create an assignment at runtime (typically, a task for a human user) associated with an open (unresolved) work item. The flow execution pauses to provide sufficient time to perform each assignment. An individual, a group, or the system (agent) must process the work item by performing analysis, data entry, and decision making to complete the assignment before the flow can progress. Assignments appear on the worklist of a user who is executing the flow, or in a workbasket. For example, you can define an assignment named "Review for Approval" so that the work item will be routed to a manager's worklist, requiring that manager to review the work item and determine whether it should be approved or not.



An assignment may include a Service Level Agreement (SLA) that defines service level goals for time intervals, escalation, and notification. If an SLA is defined, the Assignment shape includes a clock icon. If an assignment is not completed in accord with the SLA requirements, the system may raise its urgency value and may send an escalation alert. For example, an SLA can be defined for an assignment that specifies a goal date/time and deadline date/time for a manager to review and approve a work item.

Decision

A decision shape causes flow processing to continue along one path of a set of possible paths, based on input from the flow and the work item. Each connector arrow emanating from a decision shape represents a possible path the work item may take based on the decision outcome. For example, a decision shape may have two connectors that represent "Approval is Required" and "Approval is Not Required", with the choice for a specific work item based on property values in that work item.

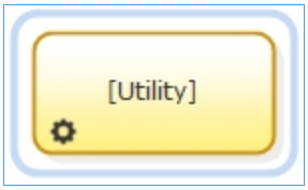
Subprocess

A subprocess shape may represent either of the following:

- A branch to another flow without return (one or more incoming connectors, no outgoing connectors).
- A call to another flow with return (one or more incoming connectors, one or more outgoing connectors).

Utility

The utility shape updates a work item without requiring human input. If a utility shape appears in the flow immediately before an assignment, the shape typically prepares a work item for user interaction; if it appears after an assignment, it usually performs post-processing of user decisions and input. For example, a utility shape can perform computations, searches, or retrieve data from a database.

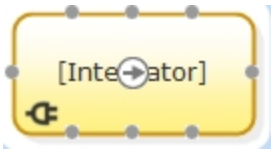


End

An end shape defines each end point within a flow. A flow may contain one or multiple end shapes. When processing reaches this shape, the work item is automatically saved, and no further processing by this flow rule occurs. If this flow was called as a subprocess by another flow, processing continues in the calling flow. Reaching an end shape does not change the work item status.

Integrator

An integrator shape allows the flow processing to connect to external systems to send and retrieve the data it needs. For example, you might use an Integrator shape to connect to a customer master database, to retrieve account balances or verify account status and values.



Connector

A connector arrow associates two shapes in a flow diagram and represents a possible path through a flow. Connectors leaving an assignment correspond to flow actions. Connectors leaving other shapes may be When conditions.

Flow Actions

Flow actions, available to users as they process assignments, allow users to capture input data and apply their expertise by making choices that advance the work item through the flow. For example, in an employee recruiting application, an employee must assess the quality of a candidate (based on a resume and application form), record his judgments and reasoning, and then choose one of three flow actions labeled Advance, Reject, and MoreInfo. These three flow actions may require different input fields and may use different forms.

There are two types of flow actions: connector flow actions and local flow actions

Connector Flow Action

The Connector shape is used to connect a shape in the flow with the next shape, to indicate a possible path for the flow execution. At runtime, users can choose a connector flow action, complete the assignment, and as a result advance the work item along that connector to the next shape.

Developers can associate a likelihood value or percentage between 1 and 100 for each connector flow action. Typically, this number is a before-the-fact opinion about the estimated percentage of times users are expected to choose that flow action at runtime.

Local Flow Action

A local flow action permits users to update assignment or work item properties, but doesn't complete the assignment. When a user chooses and completes a local flow action, the assignment remains open and on the current user's worklist; this user's input does not advance the work item through the flow. On most standard user forms, connector flow actions are listed first in the Take Action selection box, and local flow actions appear below a horizontal line.

Unlike connector flow actions, local flow actions are referenced only inside an Assignment shape (in the Assignment Properties panel), and so they have no visible representation on the flow diagram.

For example, a local flow action may allow the user to revise the mailing address, phone number, or email address of someone involved in the work item, whenever the need for such a change is discovered.

Intent-Driven Processing

Intent-driven processing allows users to easily recognize and select the appropriate flow action or other options without interpreting complex data, making complex decisions, or selecting an inappropriate option. Your application can support intent-driven processing by carefully choosing appropriate flow action names, labels, and user selection options and also by providing useful information in the assignment's Instruction field.

One best practice is to begin the Short Description field for a flow action with an active verb. The verb should describe a goal or subgoal of the flow that users would recognize as an action they can perform that will advance the work item in the intended direction. This text appears as a user choice at runtime. For example, "Approve and Submit This Order" and "Withdraw This Order Permanently" are clearer, more intent-driven than using the words "OK" and "Cancel."

Introduction to User Interface

Objectives

At the end of this lesson, you will be able to:

- Describe the concept of a user interface (UI) rule in PRPC
- Identify which rules belong to the UI category
- Use the appropriate UI rule type
- Identify which rules support the development of the UI
- Discuss basic information about:
    - Sections
    - Layouts
    - Calls
    - Labels and Fields
    - Adding and Deleting Rows
    - Columns
    - Controls

Things to Know

User Interface (UI) Concepts

We use UI rules when building portals and work forms to execute a business process. Our UI can take into account factors such as the environment and the user's domain knowledge, skills, locale and language. UI rules in PRPC are auto-generated, yet flexible to handle real-time business needs.

UI design is intent-driven; not data-driven. Thus, the business process (flow) defines the sequence of tasks and an assignment task corresponds to a user action (flow action). UI rule access is based on the access group and uses RuleSet and rule resolution parameters.

We access UI rules through the Application Explorer by expanding the User Interface category under the class name. Rules are listed by rule type, and a type only appears in the list if an instance exists in that class.

UI Rules Overview

Below is an overview of the UI rules. For more information see the UI Rules lesson.

- **Control** — Use to control how properties appear on user forms, correspondence, and other HTML forms, for both display and for accepting user input.
- **Harness** — Use to define the appearance and processing of work item forms that we use in the application to create work items and process assignments.
- **Flow Action**— Use to control how we interact with work item forms to complete assignments. After selecting one flow action, we may fill in a section of the form to complete (perform) the assignment.
- **Navigation** — Use to construct a multi-level XML document that will be used in navigation and context menus.
- **Paragraph** — Use to provide read-only text for a cell in a work item form or correspondence. Paragraphs can include bold, italics, colored text, images, links, and other rich text, plus property values.
- **Portal**— Use to create a custom portal layout for a group of users.
- **Section** — Use in conjunction with harness rules to define the appearance of work item forms, rule forms, or composite portals. The section rule defines the appearance and content of one horizontal portion of a form and the

Harness rule defines the complete form that supports all user interactions that create, update, and resolve work items.

- **Skin** — Use to identify a collection of cascading style sheets (CSS) that are used in a portal. Separate style collections apply to the workspace, work item forms, and reporting.

Overview of Sections

Sections define the appearance and content of horizontal portions of a work form and are listed in the User Interface category in the Application Explorer. They may contain properties, labels, and other UI elements.

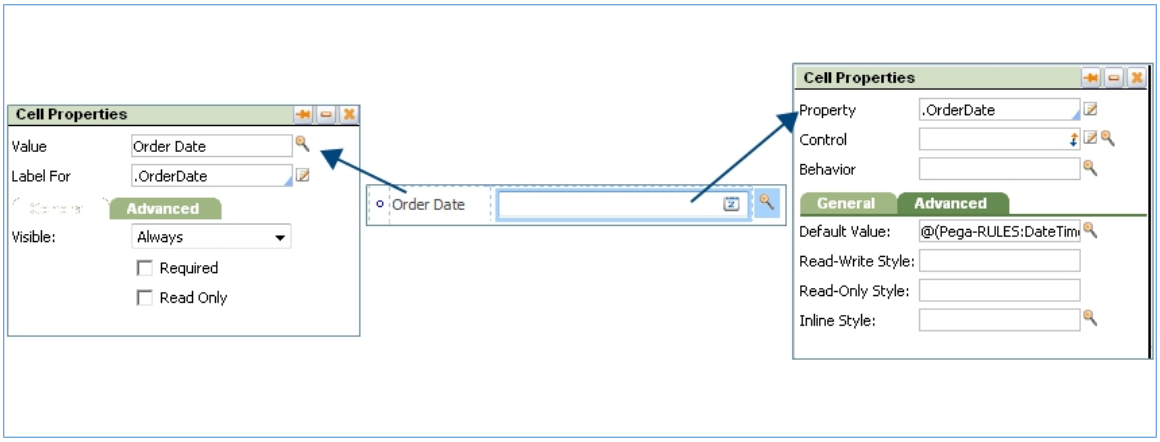
Layouts

Every section and every flow action contains at least one layout (or other form of a repeating layout). A layout defines a rectangular area on the form corresponding to the HTML <TABLE > element. The default layout in a new section is called a Smart Layout.

Smart Layouts provide templates to ease UI creation and apply a consistent look to your application. They provide default settings of single, double, or triple column pairs which consist of Label and Field columns, with separators between pairs.

Cells

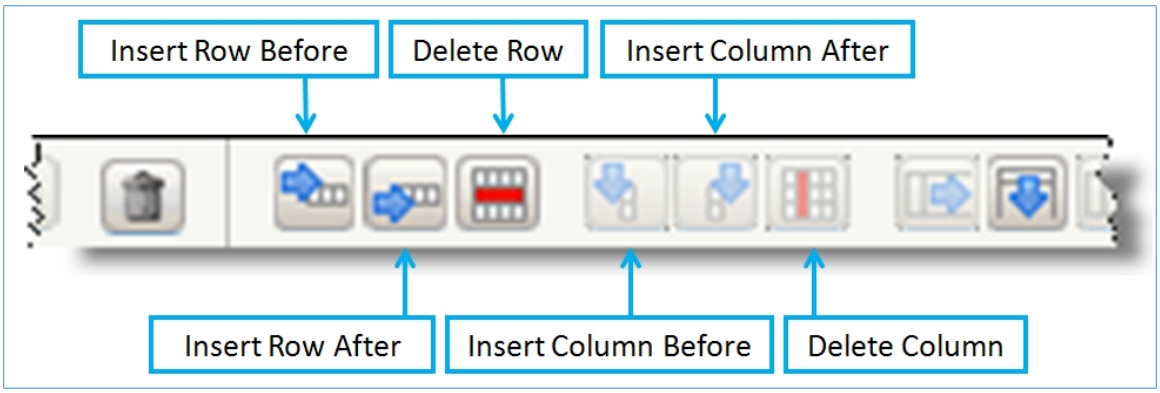
Cells are the individual components of layouts and have properties that define their look, feel, and behavior. They may contain labels, images, fields, another section, or UI controls. To modify a cell's attributes, click the cell then click the View Properties magnifying glass.



In a Smart Layout, when we add a property by dragging and dropping it into a field cell, PRPC automatically enters the label into the label cell for us.

Adding or Deleting a Row or Column

In order to add a row or column of cells, use the control icons on the Layout tab.



Module 3: Direct Capture of Objectives

Lessons Covered:

- Creating an Application Profile
- Using the Application Accelerator

Create an Application Profile

Objectives

At the end of this lesson, you will be able to:

- Explain the purpose of the Application Profile
- Review a completed Application Profile
- Describe the steps of the Application Profiler

Things to Know

When we collect use cases to guide development of a project, we need to incorporate them into the application. Then later we can directly relate each use case to the parts of the application that implement and support that use case. To record the use cases in PRPC, we use the Application Profiler to create an **Application Profile** (AP).

The Application Profiler is a guided, work-centric wizard that collects information about a project to answer the question "What will I build?" By using the Application Profiler, we can reduce the time and complexity required to gather requirements and capture business objectives, and estimate the scope of work needed.

A business architect/business analyst should run the Application Profiler as the first step in creating an application. As they proceed through the wizard, they specify details about the project, such as:

- Atomic use cases (also known as a use case)
- Requirements
- Actors
- Processes

Portions of the AP can be delegated to other team members as needed, allowing people with specific expertise to provide that expertise earlier in the development phase.

The Application Profiler can create for us:

- An **AP**, a work item used as input to the Application Accelerator. The AP is the basis for the application.
- An **Application Profile document**, that can be reviewed by project sponsors. PRPC includes a template, that we can customize, to copy data from the AP to a Word document that can then be distributed outside of PRPC.
- An **Application Profile package**, that can be used either to copy the AP to another system, or to keep as a backup.

Creating an Application Profile

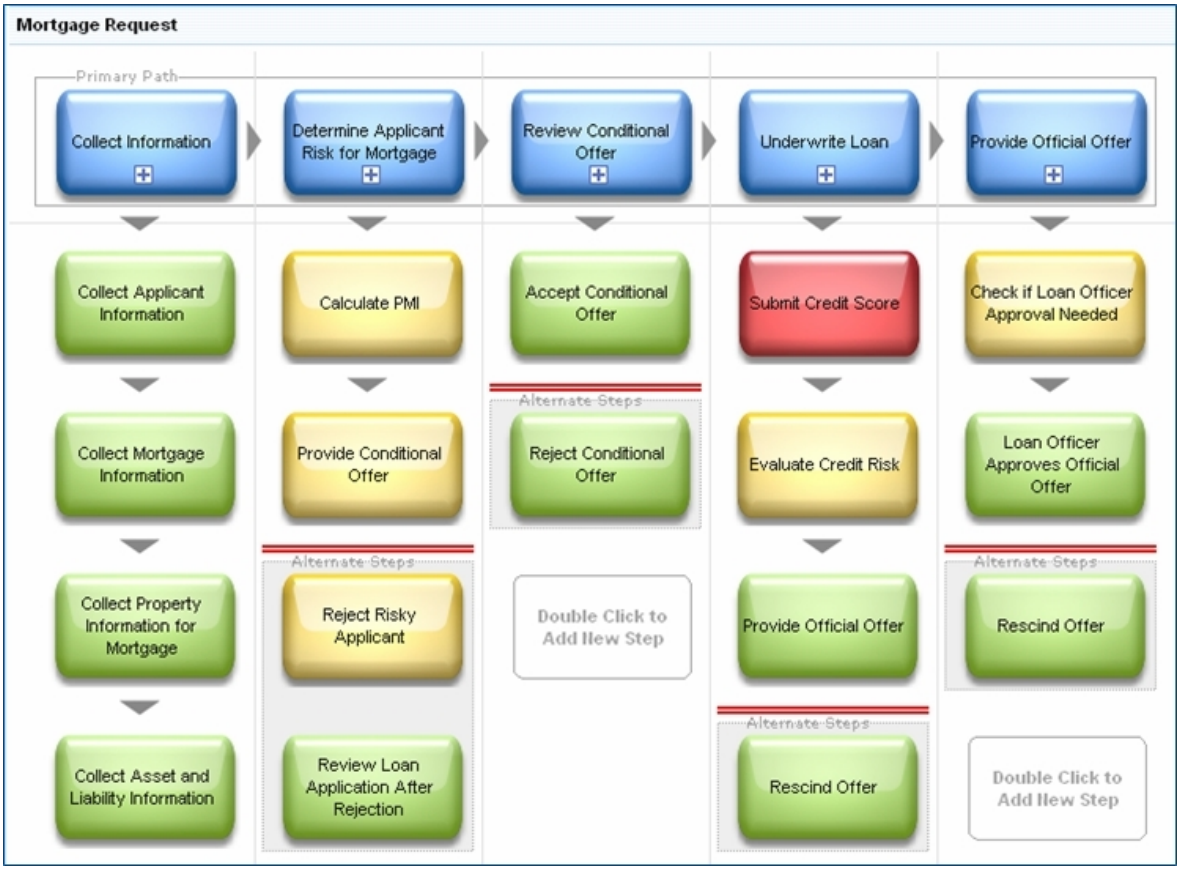
The Application Profiler guides you through a series of steps to collect the information necessary to document your application at a high level.

Overview

First, we provide a project overview. Here, we specify the basic parameters of your project, such as its name, the overarching requirements, and a brief description. This is the background that provides context for the rest of the information gathered in the Application Profiler.

Create Processes

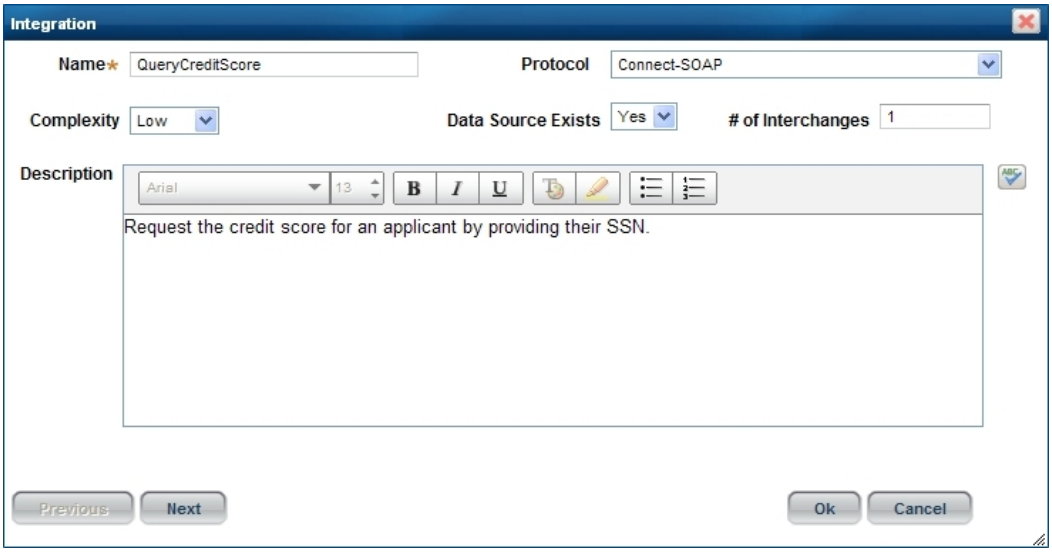
Once we complete the overview, the next step of the profiler is to complete the **Process Discovery Map**, which documents the processes that comprise the application. In this step, we document and order atomic use cases and requirements as we capture the overall business process for each work type. The best practice is to document the process as completely as possible, but the key objective is to capture, and determine the sequencing of the use cases.



For more information on the Process Discovery Map, see the *Introduction to Direct Capture of Objectives* lesson.

Interfaces

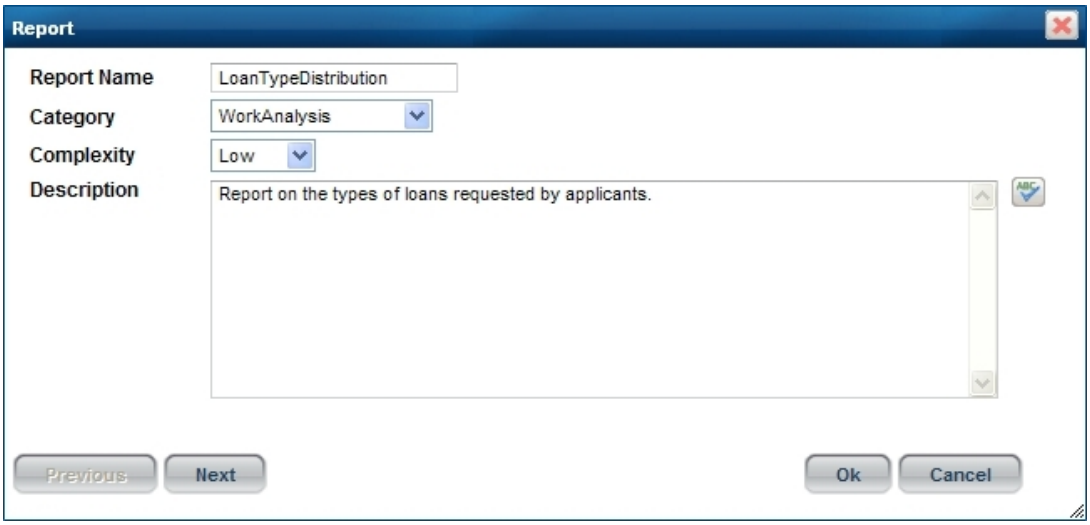
Once we document our processes, we document any connections to external data sources and external processing agents. These are the points where the application needs data from outside of PRPC, such as a credit bureau, an external customer record system, or state or national databases, plus the points where the application sends data to such external systems.



Reporting

Next, document the reports that the application should produce. Reports allow users to review work data and trends, such as:

- How many work items are currently open.
- How many work items have been resolved over the last 4 weeks.
- The cost associated with a work item.



Correspondence

Once we document the reports we want in our application, we document the correspondence that the application needs to generate. Applications can send correspondence (in email or printed form) to individuals or groups of people; we can refer to recipients either explicitly, or as a particular party identified in the work item, such as the "customer" or "originator" or "approver".

Assumptions

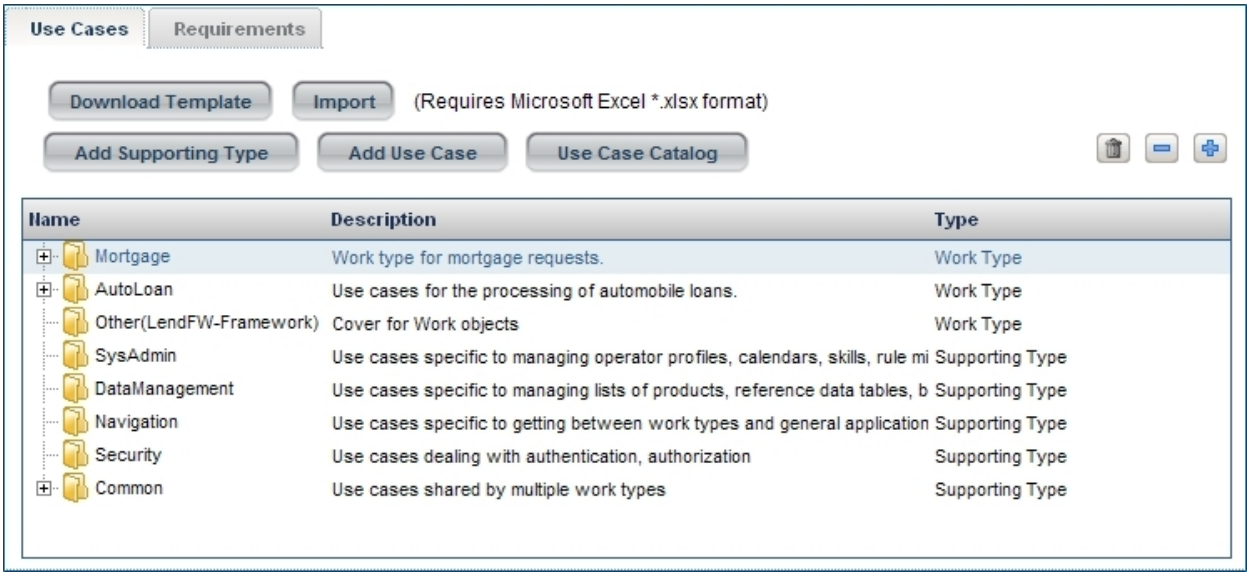
Next, we review the assumptions upon which the AP is built. These assumptions *don't* correspond to rules. Rather, they're added to the Application Profile document, creating a record of project details, such as who is responsible for providing a testing environment. PRPC provides a standard set of assumptions, to which we can add or subtract assumptions as necessary.

Participants

Next, we review the estimated project staffing and responsibilities by developer role, along with a training plan for each involved participant. We can provide details on both internal (client) resources and external (Pegasystems or partner) consultants. This allows us to identify the estimated resources necessary to implement our project.

Project Explorer

The Project Explorer is the last step in the Application Profiler, where we can review the use cases and requirements for each work type, and add new ones if necessary. In addition to the work types we've entered, the Application Profiler also lists several supporting types. These types are used to collect use cases that aren't part of the process, but are necessary to the functioning of the application, such as security and system administration tasks. We can also use the **Common** supporting type to collect use cases common to multiple work types.



Other Application Profiler Functionality

In addition to the steps we discussed above, other features of the Application Profiler are available by clicking the **Actions** menu in the upper right of the Application Profile workspace. The features are described below:

- **Actors** — displays the list of actors who will use the completed application. The names of the actors were collected from the use cases. In this dialog, we can specify how the users plan to access the application, how many of each type of user we anticipate accessing the application, and whether they're internal users, external sources, or the system itself.
- **Delegates** — allows us to create a *delegated copy* of the AP, which we can provide to other architects to provide more detail. These delegated copies can then be merged back into the main copy to create a complete AP.
- **Document** — allows us to generate an Application Profile document, which can be printed, or saved and emailed, for sharing with stakeholders and sponsors who don't have access to PRPC.
- **Overview** — displays the project overview that appeared when we first launched the Application Profiler.
- **Security** — grants access to the Application Profile to other operators. By default, only the user who created the AP can access it. We can either explicitly add specific operators, or grant access to everyone within the organization.
- **Sizing** — launches the **Project Sizing** tool, where we can diagram the estimated timeline for the project. We can also create a project sizing workbook, which allows us to enter events and estimates in more detail.
- **Package** — creates an Application Profile package, which can be saved locally. This package contains all of the information collected by the AP wizard, and allows us to move the AP to another system if necessary.
- **Save** — saves the AP with all of its current information. This allows us to continue working on the AP at a later time.
- **Withdraw** — sets the AP status to **Resolved-Withdrawn** and closes the AP. This is useful if you don't want to use this AP now. The AP can always be reopened at a later date, if necessary.

Once the AP is complete, click **Finish** on the Project Explorer step. This sets the status to **Resolved-Completed**, and completes the AP. Once the AP is closed, it can be used as an input to the Application Accelerator.

Use the Application Accelerator

Objectives

At the end of this lesson, you will be able to:

- Describe the purpose and use of the Application Accelerator (AA)
- Run the AA to create an application from an Application Profile (AP)

Things to Know

Once we run the Application Profile wizard and create an application profile, we can run the Application Accelerator to take the information collected in the AP and use it to create our application.

The AA is a guided "work-centric" wizard that collects more detailed information and decisions about the implementation of a project — information that is not collected by the Application Profiler. This information answers the question "How will I build it?" By using the AA, we can create a starting point for our application which will be a base application, compliant with best practices for application design. We can then capture any additional requirements (flow, user interface, rules, properties, etc.) in our application.

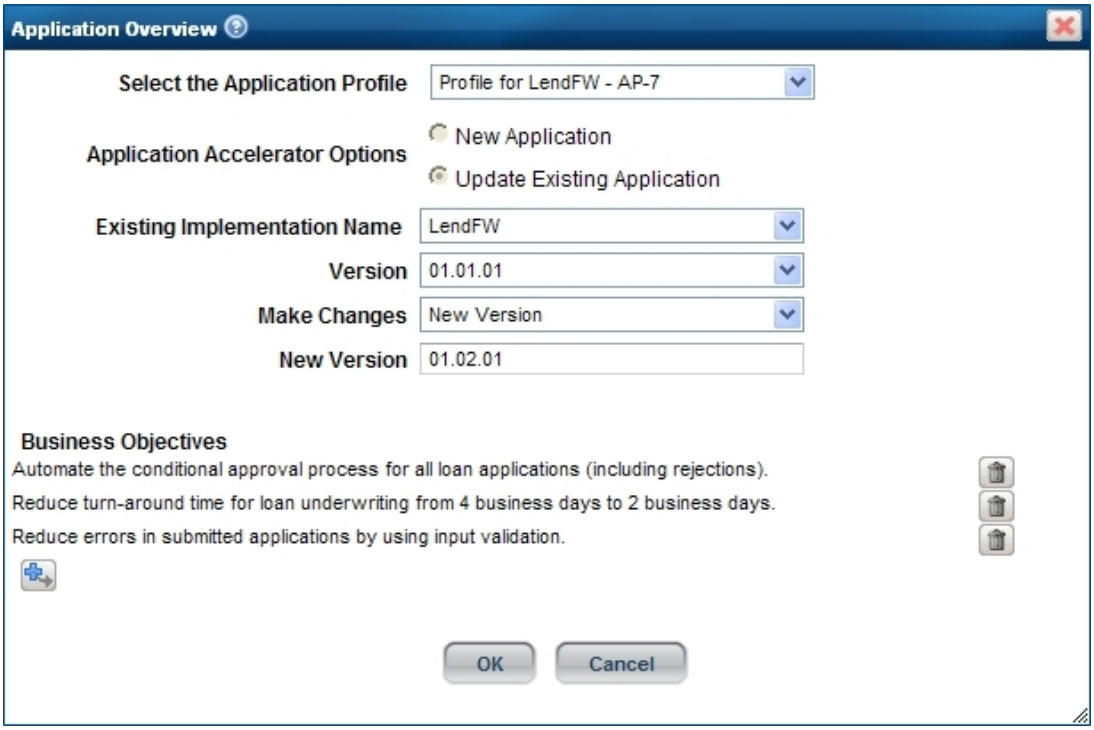
A Lead System Architect (LSA) or Senior System Architect (SSA) typically runs the AA, using the AP created by the Application Profiler. Even though System Architects don't typically run the AA, it's helpful to understand what information the wizard collects, and how it affects the application we build.

Using the Application Accelerator

Overview

First, we provide an application overview, similar to the project overview we provided in the Application Profiler. In the application overview, we can specify the AP to use as a source, and start to make decisions about the application to be built, such as:

- Whether it's a new application, or an extension of an existing application.
- For new applications, whether the application consists of a framework layer, an implementation layer, or both. (We'll discuss framework and implementation layers in the upcoming *Classes and Class Structure* lesson.)
- If we're extending an existing implementation, the name of the application (either the framework layer or the implementation layer) we want to extend.
- The name and version of the framework and/or implementation applications.



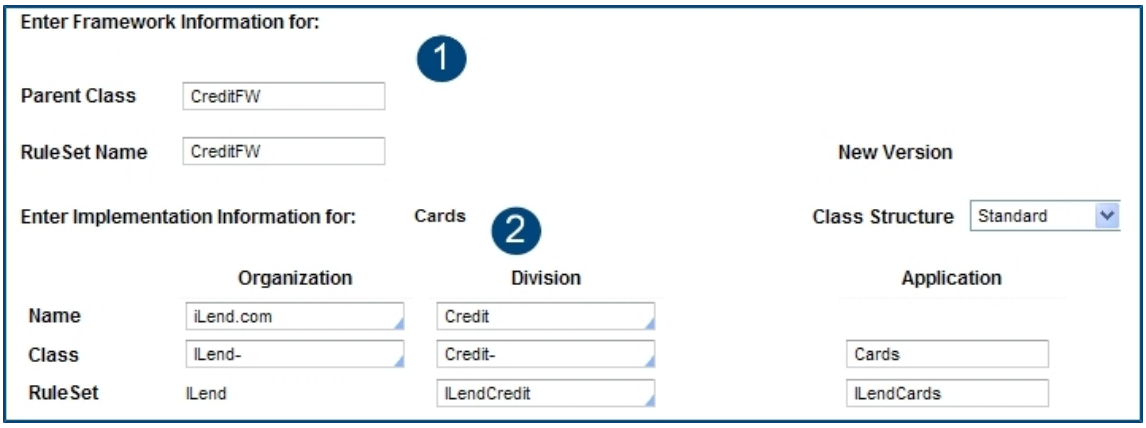
An Application Profile is not required to complete the Application Accelerator. However, the Application Profile contains information about the application, such as use cases and requirements for process steps, that can help in developing the application. As a result, **it is a best practice to complete an AP before running the AA**.

Note: The selections we make here — including those inherited from the AP — affect the order of steps in the Application Accelerator, as well as the options available on some steps.

Base and RuleSets

After we complete the overview, the first step of the Application Accelerator is to name the RuleSets that the AA will create for the application.

In this step, we specify the classes and RuleSets to be created by the AA to collect the rules that make up our application. In addition, the organization and division names are used to create the corresponding organization structure records. (For more information on how PRPC uses an operator's organizational structure, see the *Organizational Structure* lesson.)



The appearance of this form changes, depending on the options chosen in the overview.

1. The **Enter Framework Information for** section appears only if the application will have a framework layer.
2. The **Enter Implementation Information for** section always appears, but if the application will not have an implementation layer only the **Organization** column is available. This occurs because organization information applies to both the framework and implementation layers.

If we're extending an existing application, this form instead prompts for a new RuleSet version to contain the rules that make up this new application version.

Create Processes

The Create Processes step performs the same function as its counterpart in the Application Profiler. This step exists in the AA to allow users who bypass the Application Profiler an opportunity to create a Process Discovery Map.

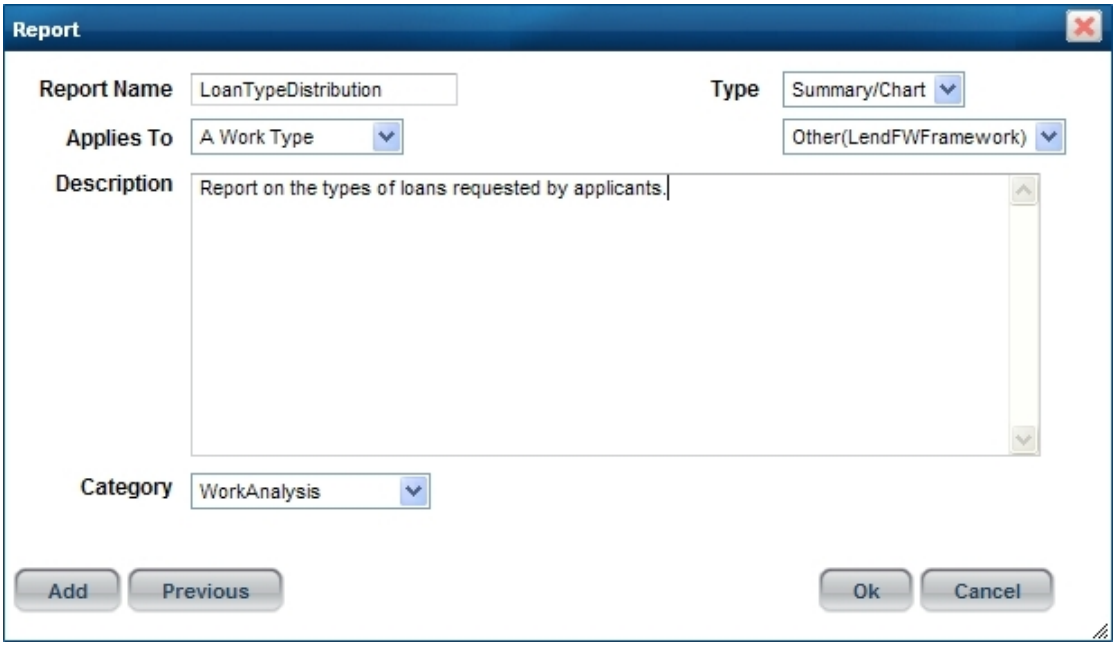
Class Structure

After reviewing our processes, we can advance to the **Class Structure** step. Here, we can organize our work types into a hierarchy. This allows us to group common work types under a shared layer, creating an additional opportunity for reusing rules. Consider an application to process requests for auto loans, home equity loans, and mortgages. Since home equity loans and mortgages both involve real estate property, we might find that some rules are the same for these two loan types, but the rules do not apply to the auto loan type. As a result, we might consider grouping mortgage requests and home equity loan requests under a shared layer, where we can collect rules that are common to these two work types.

We can also provide a parent for each work type, by selecting a value in the **Inherits from** drop-down. This parent provides basic rules to govern how the objects created for each work type will behave. This option is only available for new applications. If the application is instead extended, only the new work types can be set; existing work types cannot be changed in the AA.

Reporting

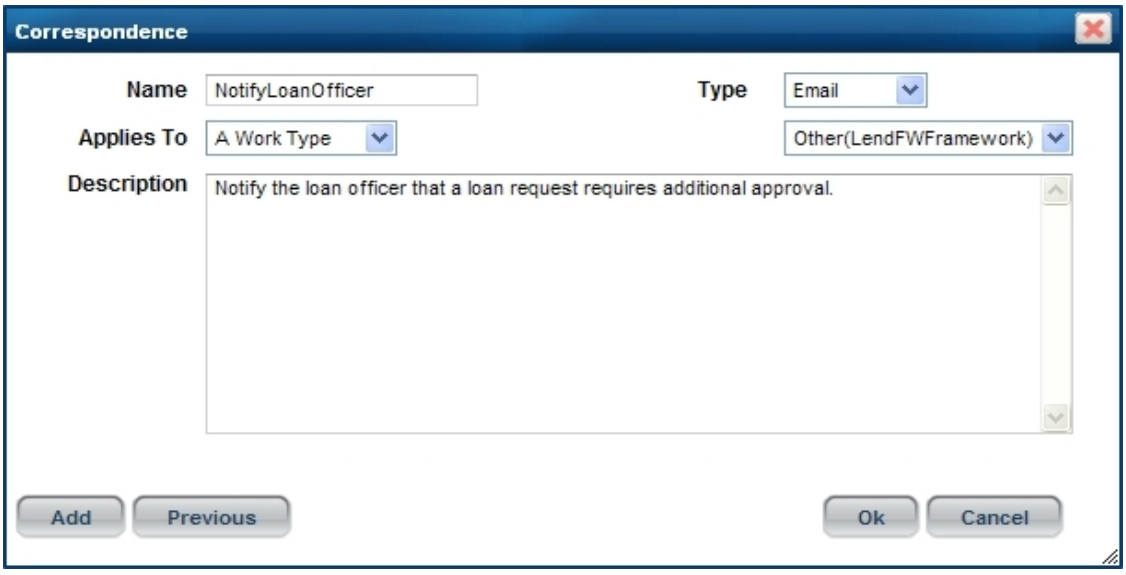
Once we create a class structure, the AA advances to the **Reporting** step. Here, we can elaborate upon the reports we defined in the AP, specifying the type of report we want — either summary or list. We can also specify how the report should be applied — to a work type, to a shared layer, or to assignments. The AA provides additional options for refining reports, which allows us to create the report in a specific work type, a shared layer, or the root layer (also called the class group). For reports applied to assignments, we can restrict the report to assignments in worklists, or workbaskets, or include both.



We can also choose a standard report from the report catalog, rather than create a new report.

Correspondence

Once we define the reports we want to create, the AA advances to the **Correspondence** step. Here, we can elaborate on the correspondence we defined in the AP. We can choose the type of correspondence to send — Email, Fax, Mail or PhoneText (text messages or SMS) — and whether the correspondence should apply to either a shared layer or a specific work type.



Roles

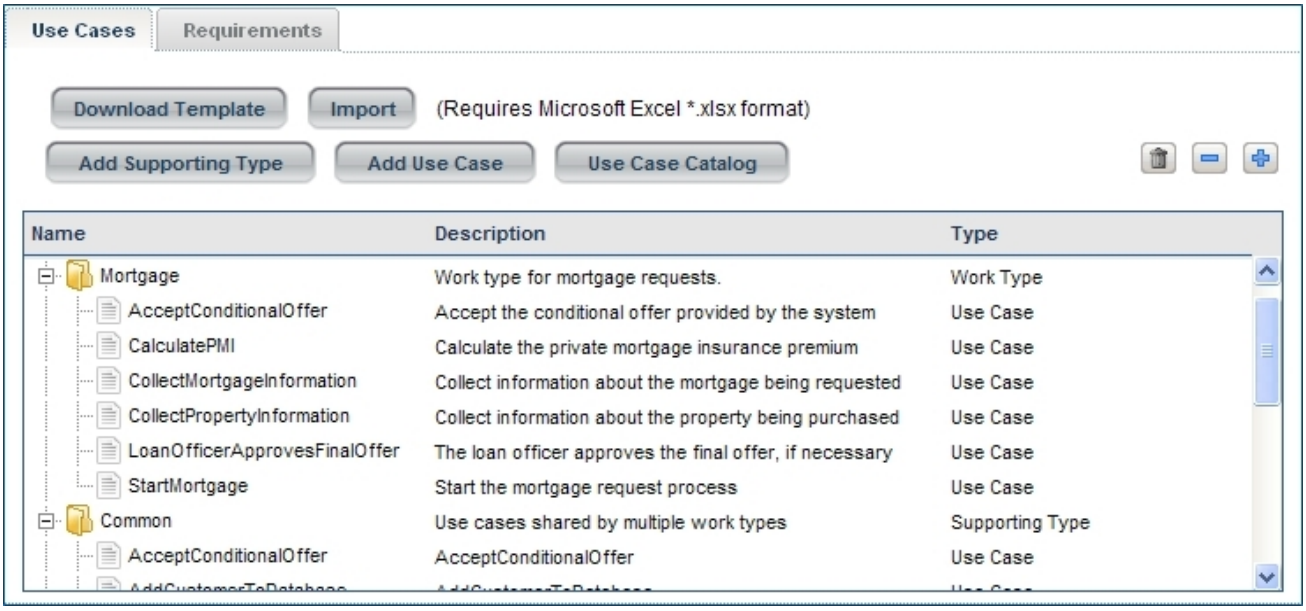
Here, we create a list of work party roles for the application. We can specify one of the standard roles, such as Operator, Manager, or Interested, and associate each role with the one of the human actors specified in the AP. We can also opt to use the operator entering the work as the default operator for the role. Typically, this happens with the Originator role. We can also allow more than one occurrence of a party role in a work item. For example, a work item may have several interested parties.

Other Extensions (Application Extension only)

This step is available when we create a new application built upon an existing framework. The AA lists existing rules that can be copied over to our new application. By default, these rules are selected, meaning they will be copied to the application. If we do not want to copy a rule, we can deselect it.

Project Explorer

The Project Explorer step performs the same function in the AA as it does in the Application Profiler, allowing us to review, modify, and/or delete work types, use cases, and requirements. The supporting types and their uses cases also appear in this step.



Review Application

This is the final step of the AA. Here, the AA lists the rule instances to be created, and identifies in what RuleSet each one will be created. We can also review the class structure the AA will create.

Other Application Accelerator Functionality

In addition to the steps we discussed above, other features of the AA are available by clicking the **Actions** menu, found in the upper right corner of the AA workspace.

- **Actors** — displays the list of actors who will use the completed application. The names of the actors are collected from use cases. In this dialog, we can specify how the actors plan to access the application, how many of each type we anticipate will use the application, and whether they're internal users, external sources, or the system itself.
- **Overview** — displays the project overview that appeared when we first launched the AA.
- **Save** — saves the AA work item with all of its current information. This allows us to resume our work at a later time. Unlike the AP, however, in-process AA work items aren't available from the **Application** menu. To open an in-process AA work item, click the Pega button and use the **All Wizards** tool available from the **Application** category.
- **Withdraw** — sets the AA work item's status to **Resolved-Withdrawn** and closes it. This prevents other users from creating an application with this AP. is useful if you decide you do not want to use this AA but want to save it. The AA work item can be reopened at a later date.

Once the AA work item is complete, click **Build this Application** on the Review Application step to create the application. Once PRPC creates the application, we can click **Switch to this application** to switch to the new application without logging out and back in.

Module 4: Managing Rules

Lessons Covered:

- Rules and RuleSets
- Classes and Class Structure
- Using RuleSets
- Rule Resolution

Rules and RuleSets

Objectives

At the end of this lesson, you should be able to:

- Explain the difference between rule types and rule instances
- Explain the difference between RuleSets and classes
- Learn how to reuse rules to decrease development time and increase application quality

Things to Know

Common PRPC Terminology

Rule Type

A **rule type** is a data template that defines the user interface, business logic, process, or data behavior within PRPC.

Rule Instance

A **rule instance** is a specific occurrence of a rule type, just as a work instance is the specific occurrence of a work type. If we save a flow rule, we now have a specific instance of the flow rule type.

RuleSet

A **RuleSet** is a container for rule instances. RuleSets allow us to bundle rules for reuse. Applications are really just stacks of rules, applied in a specific order. RuleSets are further subdivided into RuleSet versions, allowing us to revise rule instances as we use them. We'll discuss RuleSet versioning shortly.

When we need to migrate an application from one system to another, RuleSets make that process easier for us. We'll see this later when we discuss application migration.

Class Instance

A **class instance** represents the applicability of a rule, which is often referred to as the *scope* of the rule. Class instances — commonly called classes — do not *contain* rules, as RuleSets do. Rather, class instances determine how the rules contained within the RuleSets *are applied*.

For example, consider a flow rule — the rule instance is *contained* within RuleSet X, but *applied* to class Y. This means that any application that uses RuleSet X, and references class Y, can use the flow. If either the RuleSet or the class is missing, the rule is not visible to the application, but for different reasons.

1. If the **RuleSet isn't part of the application**, the flow *doesn't exist within the context of the application*.
2. If the **application doesn't inherit rules from the class**, the flow *can't be found*.

One easy way to remember the distinction: classes are instances of *class rules*, and rule instances must **always** be defined within a RuleSet.

Application

An **application** is a collection of one (or more) work types. From the end user's perspective, an application is defined by what they can do with it, such as: open a bank account, process a change of address, or create a purchase request.

The end user's concept of an application differs from the application developer's concept, which revolves around the application rule type. We'll discuss the application rule type later in this lesson.

Naming RuleSets

When referring to a RuleSet — such as to save a rule instance — you must specify two specific pieces of information: a **name**, and a **version**, separated by a colon; for example, YourLoanCo:01-03-05.

RuleSet names are limited to a maximum of 64 characters, consisting of either mixed-case letters, numbers, dashes ("-"), and underscores ("_"); spaces are not permitted. When naming a RuleSet, use a name that correlates with the class(es) to which its rules will apply. The Application Accelerator does this automatically when it creates the class structure and RuleSets for an application.

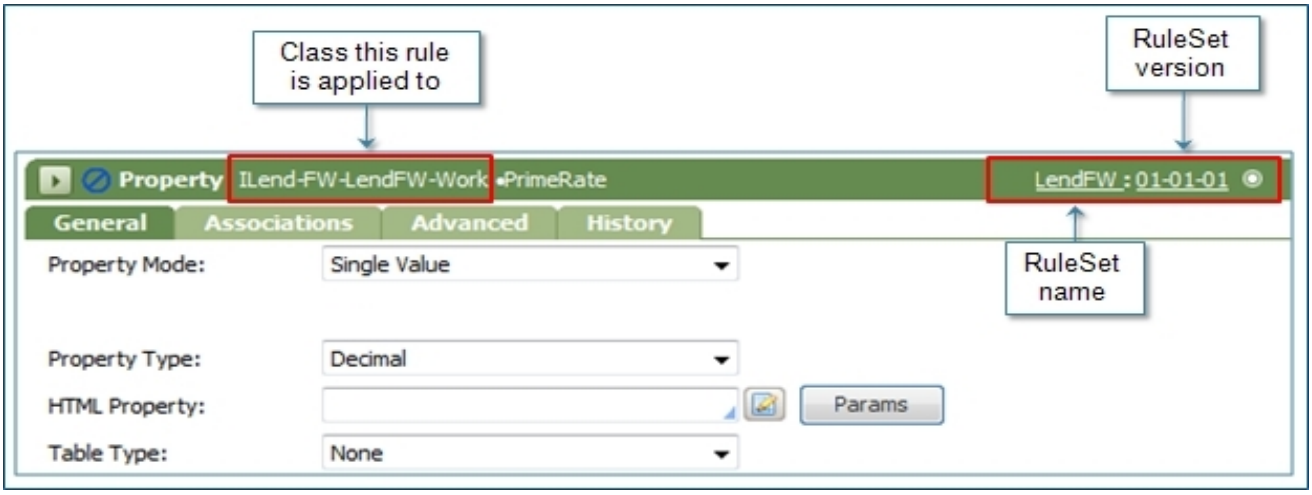
Each RuleSet is made up of one or more RuleSet versions. RuleSet versions are the actual rule containers. This allows us to revise the rule from one release to the next, without losing the earlier instance(s) of the rule. **RuleSet versions consist of three two-digit numbers**, separated by dashes, in the following order:

- **Major version** — The initial or subsequent substantial release of an application. A major version change encompasses extensive changes to functionality.
- **Minor version** — An interim release, usually updates to the product with new are functionality added; there should be no upgrade issues for data. Use a minor version to isolate ongoing development and customization from the locked master RuleSet.
- **Patch version** — An interim release, usually bug fixes and interim changes. There may be several revisions within a minor version release (05-01-10, 05-01-50, 05-01-53, etc.)

As a **best practice**, use only the major and minor version numbers when specifying a RuleSet version from which to read; PRPC automatically assumes the highest patch version.

Identifying Class and RuleSet Membership

Almost every rule instance in PRPC indicates its RuleSet (including version) and class (some exceptions are application rules, HTML fragments, and binary and text files). This allows us to quickly determine the usability of the rule. For example, the property **PrimeRate** is defined in RuleSet version **LendFW:01-01-01**, and applied to the **ILend-FW-LendFW-Work** class. This means that any rule defined in the 01-01-01 or later version of the LendFW RuleSet, and applied to the ILend-FW-LendFW-Work class or any of its descendents, will be able to use the PrimeRate property.



RuleSet Prerequisites

By using RuleSet prerequisites, we can keep individual RuleSets smaller and more focused. This makes our RuleSets more reusable.

How Rule Prerequisites Work:

1. RuleSet B lists RuleSet A as a prerequisite.
2. When saving RuleSet B, PRPC verifies that RuleSet A — the prerequisite — exists.
3. When migrating RuleSet B, either the product archive or the target system must include RuleSet A.
4. If RuleSet C lists RuleSet B as a prerequisite, then RuleSet C automatically gains access to the rules in RuleSet A, as well.

As a best practice, our RuleSet hierarchy should mirror our class structure and its inheritance model. We'll learn about inheritance in the in the **Classes and Class Structure** lesson.

Why Use Rule Prerequisites?

Rule prerequisites allow for the reuse of existing rules. When the Application Accelerator creates the RuleSets for an application (referred to as the *RuleSet stack*), the base RuleSet always lists the most recent version of the **Pega:ProcessCommander** RuleSet. This gives the application access to all of the properties, UI, and other rules provided by PRPC.

By using RuleSet prerequisites, developers can create highly focused, reusable RuleSets. Consider an application that processes account openings. Each type of account — checking, savings, and investment — has specific attributes. However, there are always certain attributes common to all accounts, such as an account number and the account holder's name. These common elements can be saved in RuleSet A, which would then be listed as a prerequisite for the RuleSets specific to each work type. This way, we can reuse the common elements in each work type, without the need to copy them into each RuleSet dedicated to a specific work type.

Application Rules

Application rules represent the developer's concept of the application, as opposed to the user's concept. An application rule defines a set of functionality by collecting RuleSets into one reusable entity.

One of the main functions of the application rule is to aggregate RuleSets for reuse. Application rules don't always correspond to user applications, though they *do* correlate somewhat. The work types that a user interacts with, and the processes they would run to resolve work assigned to them, are defined within RuleSets, which developers can bundle into an application by using an application rule. An application rule can be used as the foundation of another application rule, allowing developers to more easily integrate functionality into applications.

Let's return to our application that processes account openings. Each type of account opening — checking, savings, and investment — involves the same basic process: collect information about the account holder, generate an account number, and reserve that account number in a system of record. Rather than independently creating three separate applications — one each for checking accounts, savings accounts, and investment accounts — it's more efficient to create **four** applications: one general-purpose account opening application, then one application for each account type. This way, we focus our development effort on creating one robust foundation application (the general- purpose account opening application), and build the account-specific applications upon it. This allows us to automatically reuse common elements in the three account-specific applications.

This is the rationale behind dividing an application into *framework* and *implementation* layers. We'll discuss these layers in the **Classes and Class Structure** lesson.

Classes and Class Structure

Objectives

At the end of this lesson, you will be able to:

- Reuse existing rules
- Review the inheritance hierarchy for an application

Things to Know

Application and Class Structure

An application is organized as a hierarchy of classes. This hierarchy is referred to as a **class structure**, and it governs how individual rules are applied.

The class structure is organized from most specialized to most generalized, and includes all of the classes that make up the application:

- Implementation classes created by the Application Accelerator
- Framework classes created by the Application Accelerator
- Pega Framework classes, such as Customer Process Manager™ (CPM)
- Standard classes provided in PRPC

Classes

In PRPC, classes can be either **abstract** or **concrete**.

Abstract classes cannot be used to create a work item. Instead, abstract classes are used to collect rules shared by one or more work types (which may exist in multiple applications). When referring to an abstract class, always include a trailing hyphen, such as ILend-FW-.

Concrete classes *are* used to instantiate work items, and are indicated by the lack of a trailing hyphen.

Inheritance

The main purpose of the class structure is to promote the reuse of rules through a mechanism called **inheritance**. When a class is said to inherit from another class, the inheriting class (referred to as a *child* class) gains use of all of the rule instances applied to the class from which it inherits (referred to as a *parent* class).

Consider an application to process travel reservations. A user can purchase a plane ticket, make a hotel reservation, and rent a car — and combine them into one itinerary. Each of these transactions — flight, hotel, car rental — corresponds to a different work type, with a different process. However, these transactions share one common element; the purchaser. Rather than create three separate forms to collect information about the purchaser — one for each work type — it makes more sense to create the form in one class, and have each work type inherit from that class.

To override an inherited rule, you need only to create a new rule of the same type and name, and apply it to the child class.

PRPC Inheritance Model

PRPC uses a dual-inheritance model. Dual inheritance is conceptually similar to single inheritance — except that it involves *two* pathways to inherit rules.

The first type of inheritance in PRPC is **pattern inheritance**. Pattern inheritance is optional — though it is enabled for most classes, including those created by the Application Accelerator — and involves parsing class names. In PRPC, the hyphen character delimits class names; to determine the parent of a particular class using pattern inheritance, just look at the characters to the left of the right-most hyphen.

For the class **TravelCo-FW-BookingFW-Work**, the pattern inheritance pathway would be:

- **TravelCo-FW-BookingFW-**, which inherits from
- **TravelCo-FW-**, which inherits from
- **TravelCo-**

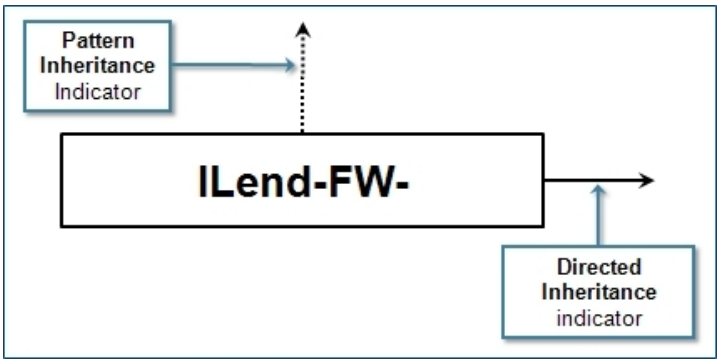
Remember that abstract classes are indicated by use of a trailing hyphen.

The second type of inheritance in PRPC is **directed inheritance**. Directed inheritance is mandatory, and *must* be specified on the rule form for a class instance. In most applications, including those created using the Application Accelerator, classes are configured so that during rule resolution, pattern inheritance takes precedence over directed inheritance.Since best practices dictate that our class structure should align with our organization structure, directed inheritance is the preferred method to access rules in standard PRPC classes such as **Work-**, **Work-Cover-**, and **@baseclass**.

PRPC determines the inheritance independently for each class. As a result, the inheritance pathway may differ significantly for two classes in the same application.

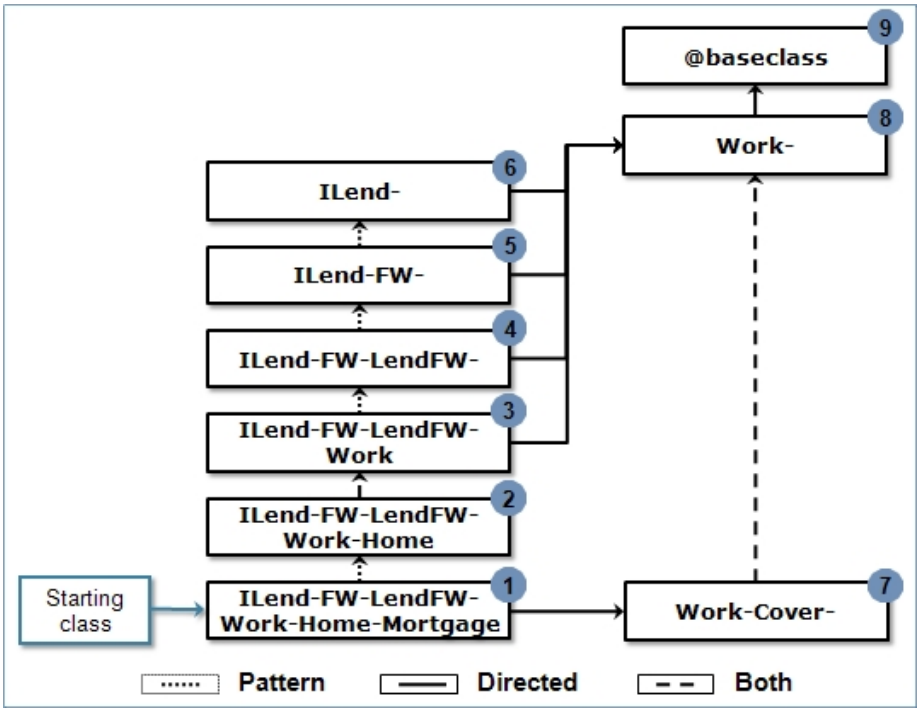
Reviewing Inheritance

When reviewing class inheritance, PRPC indicates pattern inheritance with a dotted line, and directed inheritance with a solid line.



You can review the inheritance pathways in your application with either the Class Structure viewer or the Inheritance viewer. To use either or these tools, right-click the class in the Application Explorer and select either **Structure** or **Inheritance**.

The following is an example of how inheritance might work in an application. The number next to each class is its order in the inheritance hierarchy.



Work Classes and Class Groups

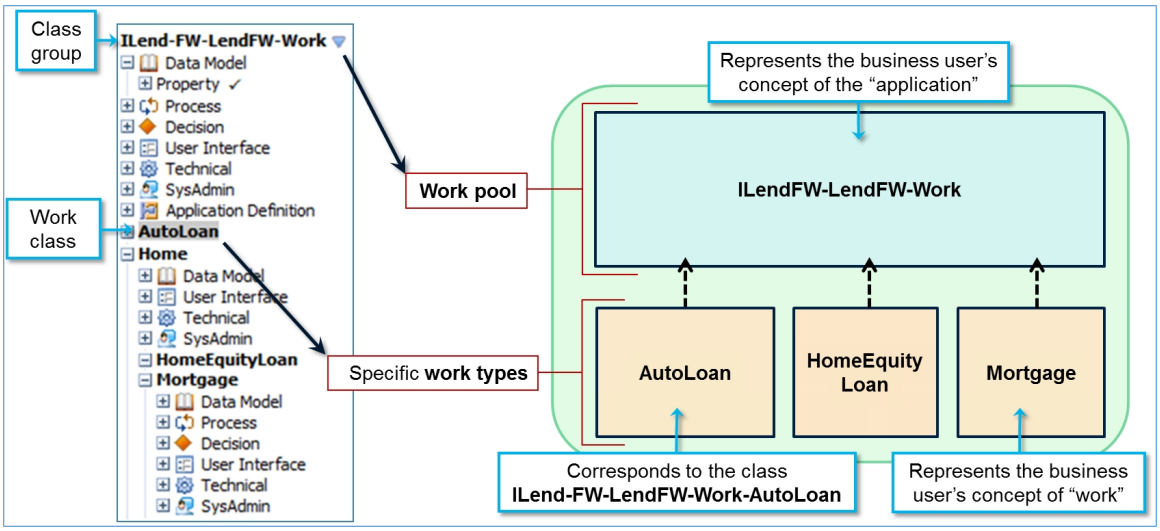
Work Class

In an application, a **work class** is a class that corresponds to a *work type*. The work class can be instantiated — so it must be a concrete class — and contains rules specific to that work type, such as flows, properties and sections. Rules defined in the work class override rules defined in any parent class.

Class Group

The **class group** is a class above the work class, which allows you to tie together multiple work classes. As a result, class groups must ultimately trace their inheritance back to the **Work-** class. Rules defined in the class group are inherited by any of its work types; this allows you to collect common rules in the class group, rather than implement them in each work type. In addition, the class group defines the database table in which PRPC stores work item data for the child work types.

The class group is analogous to what users think of as the application. The class group is equivalent to the more business-friendly term *work pool*.



Reusability

Reusability is one of the critical aspects of PRPC application development, and is almost always integral to the success of a project. As we have discussed previously, rule reuse allows you to create a rule once, and use it in multiple work types — or even applications. Reusability is the reason for inheritance, and is the primary consideration when determining the class structure for an application.

PRPC supports two types of reusability: **application**, and **enterprise**.

With **application reusability**, the focus is on aggregating common functionality shared by multiple business units. For example, an insurance company might underwrite policies across multiple lines of business, but each policy type requires some sort of policy administration process. In this case, the policy administration functionality would be aggregated into one common application, and reused by each business unit administering policies.

With **enterprise reusability**, the focus is on aggregating common functionality shared across the entire business. Enterprise reusability typically focuses on low-level functionality, such as LDAP integration.

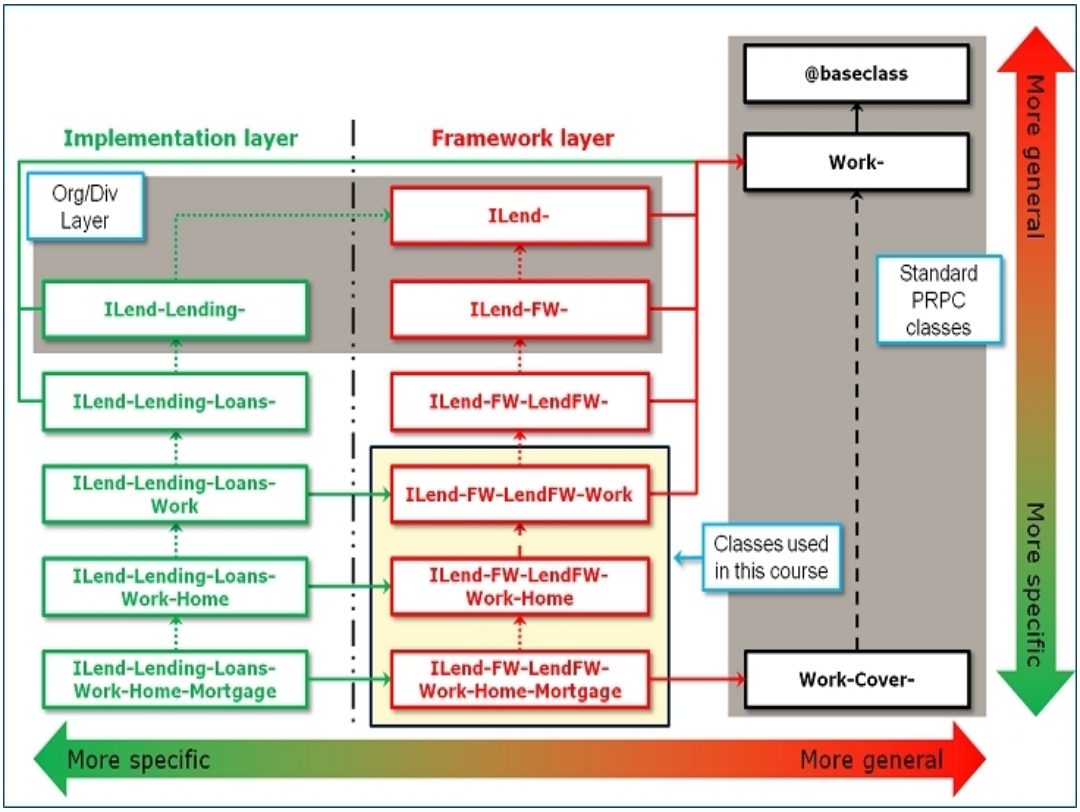
Framework and Implementation Layers

Most enterprise applications are built in two layers: a reusable **framework layer**, and a non-reusable **implementation layer**. The framework layer is used to create generalized, *reusable* applications, which can be used as the basis for more-specific application. The implementation layer is used to create a *specialized* application.

A framework application can be used as the basis for multiple implementation applications, but an implementation application can only be built on one framework application. However, framework applications can be nested to aggregate functionality.

If an insurance company wanted to standardize its policy administration across multiple business lines, they could create a generic policy administration application in the framework layer. This generic application could be used as the basis for policy administration applications that are tailored to the commercial property, personal property, and personal automobile business lines.

The application we use in the remainder of this course utilizes the following class structure.



Using RuleSets

Objectives

At the end of this lesson, you will be able to:

- Describe how PRPC controls access to RuleSets and rule instances
- Identify the RuleSets available to an user
- Determine the order in which RuleSets are prioritized for an user

Things to Know

RuleSet Uses

The primary uses for RuleSets are listed and described below.

Migration

Since RuleSets are containers of rule instances, we can migrate rules from one system to another. This allows us to move an entire collection of rules at once, rather than individual rule instances. (Though there are situations where moving a *particular* rule instance is preferable.)

Rule Versioning

RuleSets are subdivided into RuleSet versions, which allows you to iterate RuleSets as you update applications. By using versions, we can update the rules that make up our application, while keeping the prior versions available for fall-back.

Access control

We can restrict rule access to specific users. This can be desirable in several situations when:

- We're editing a rule instance, and don't want other architects to overwrite our changes.
- We've finished updating the contents of a RuleSet version, and want to freeze the version and move development work to a new version.
- We're releasing a new version of an application, and need to train employees on this new version. Since we don't want anyone to use the new version until they've been properly trained, we need to restrict people's access to the new rules in the application.

In prior lessons, we discussed the first two uses (migration and rule versioning). In this lesson, we'll focus on access control.

RuleSets and Rule Access Control

In PRPC, access can be granted up to and including the listed major, major-minor, major-minor-patch versions. An user's **operator profile** displays the RuleSets (including the highest version) that they can access.

As new RuleSet versions are layered on top of earlier versions, the rule instances in the new versions override previous rule instances. Think of a RuleSet version as a transparent grid of squares, with each square representing a rule instance. As we update rule instances in the latest RuleSet version, we color in that rule's square, so we can no longer see through to the previous version(s).

We can control an user's ability to access RuleSet versions. This is most frequently done on the application rule, where we specify the RuleSet versions (and their predecessors) that make up the application.

Rule Check-out/Check-in

Rule check-out allows an user to restrict edit access to a particular rule instance. This prevents other users from updating the same instance at the same time, and avoids conflicts when saving the rule, which would otherwise cause changes to be unknowingly overwritten. Rule checkout is enabled for a *specific RuleSet*; by default, this option is enabled for all RuleSets created by the Application Accelerator.

When rule check-out is enabled, each user has access to their own *personal* RuleSet. This personal RuleSet — identified as **[OperatorID]:01-01-01** — contains all of the saved rule instances checked out by the user. When the user checks in a rule instance, the instance is updated in the application RuleSet and deleted from the user's personal RuleSet. For users who cannot check out rules, this RuleSet remains empty. If rule check-out is disabled, PRPC does not create personal RuleSets.

Locked RuleSets

Once a development team completes its work on a particular RuleSet, that RuleSet is transferred, or *released*, and migrated to another environment (such as a testing environment, and ultimately to a production environment).

To avoid changes to the contents of the RuleSet, the RuleSet can be **locked**. When a RuleSet is locked, the contents of the RuleSet cannot be checked out and/or edited. This prevents inadvertent rule changes — such as deletions — that can affect an application's functionality.

RuleSet List Generation

Each user has a distinct collection of RuleSets available to them, from which PRPC attempts to locate rule instances when needed. This collection is referred to as the **RuleSet list**. Let's discuss how PRPC generates and uses an user's RuleSet list.

An user gains access to a RuleSet through membership in an **access group**. Access groups have several uses, such as specifying:

- Which portal an user sees upon accessing PRPC
- The work pool available to the user
- The application available to the user

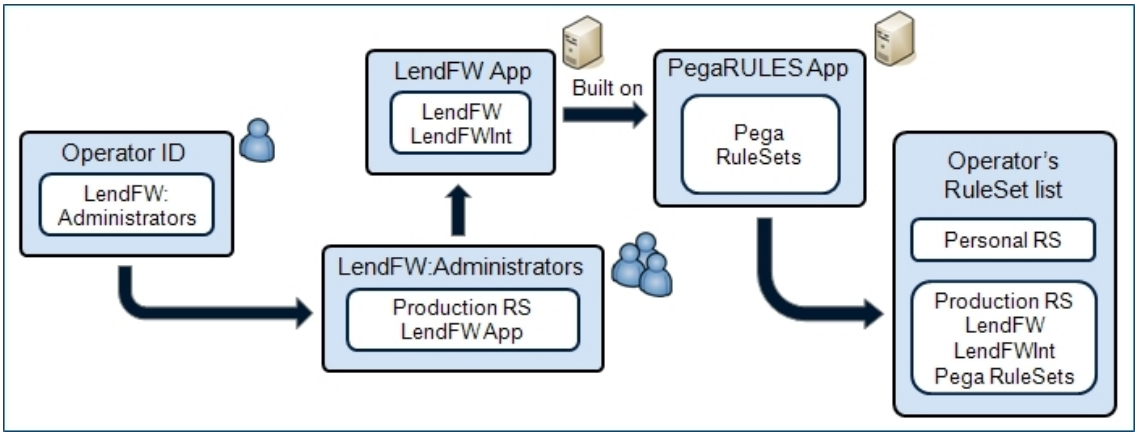
For this lesson, we're only concerned with the access group's impact on rule availability. Let’s explore how this works:

- A *system administrator* assigns the user to one or more **access groups**. If a user is a member of multiple access groups, one must be designated as the *default* access group. Only the default access group is active during the user's session.
- The access group references an **application rule**, which specifies the RuleSets that members of the access group can access. This application rule may reference another application rule, which may access yet another application rule. Eventually, an application rule will reference the PegaRULES application, which provides the user with access to the standard rules provided by PRPC.
- The access group may optionally list **production RuleSets**. These are unlocked RuleSets on production systems, generally used to store rules that need to be updated after the application has been released into production. One example of such rules are reports, which can be created as needed by managers.
- Additionally, each user's **personal RuleSet** supersedes all of the other RuleSets. This allows the user to review the effect of any rule changes they make.

At the end of this process, PRPC assembles the user's RuleSet list in the following order:

1. [OperatorID]:01-01-01 (personal RuleSet)
2. Production RuleSets (open RuleSets on production systems)

3. Application RuleSets
4. RuleSets from referenced applications
5. Standard PRPC RuleSets



The order of the RuleSet list is important, as the contents of a RuleSet higher on the list overrides the contents of any RuleSets below it on the list.

The default mechanism for assembling the RuleSet list is the user's assigned access group. An access group is usually specified on the user's Operator ID record. If an access group is *not* listed there, PRPC uses the user's organizational hierarchy to check the organization and division records for access groups.

Rule Resolution

Objectives

At the end of this lesson, you will be able to:

- Describe the rule resolution process
- Discuss how class inheritance affects rule resolution
- Demonstrate how to control the availability of rule instances

Things to Know

What is Rule Resolution?

When we run a process, PRPC uses the various rule instances we've built in our application. But how does PRPC know which rule instance to use? The answer is a process called **rule resolution**.

As we build an application, we'll create specialized rule instances. In the case of an approval action, we might have the standard Approve flow action in Work-, a specialized version in a work type, and a generalized version in the class group. PRPC determines the one to use with its rule resolution algorithm.

Rule Resolution and Class Inheritance

The most easily understood part of the rule resolution algorithm is the inheritance component. Using the previous example, PRPC identifies the candidate Approve flow actions in the following order:

1. Work type — Most specific
2. Class group — usually reached by pattern inheritance
3. Work — reached by directed inheritance
4. @baseclass — Most general

However, there are other factors that can affect which rule instance is used. In this lesson, we explore rule resolution in more depth.

Rule Availability

One of the factors that affects rule resolution is rule availability. There are five options for rule availability in PRPC. The first two options expose a rule instance during processing, while the remaining three prevent its use. Each option is described below:

1. **Yes** — The rule instance is always available, and can be copied without restriction.
2. **Final** — The rule instance is always available during processing, but cannot be copied into another RuleSet without using a new name. "Final" rule instances are just that — the end of the line.
3. **No/Draft Mode** — The rule instance is ignored; PRPC skips the rule instance and proceeds to the next one in the hierarchy, starting with the next-highest RuleSet version.
4. **Blocked** — The rule instance is ignored; when PRPC encounters a blocked rule, it immediately stops the rule resolution process, and returns an error to the user.
5. **Withdrawn** — The rule instance is ignored; when PRPC encounters a withdrawn rule, it skips any earlier version in the same class, and proceeds to the next-highest class in the hierarchy.

Blocked rules and withdrawn rules are both invisible to rule resolution. Similarly, both blocked rules and withdrawn rules prevent lower-version rules with the same RuleSet and visible key from being selected by rule resolution. However, a

blocked rule may block other rules in **any** RuleSet, and a blocked rule stops rule resolution from finding rules in higher classes. A withdrawn rule affects other rules only in **one** RuleSet and class. Put another way, withdrawn rule instances prevent execution of all rule instances *in a particular class*, while blocked rules prevent execution of the rule *no matter what class they are in*.

Withdrawn rules are typically used to "delete" a rule in a locked RuleSet version. For example, if you initially built an application with the Approve flow action in the work type, and decide three releases later to generalize the Approve action in the class group, you would need a way to prevent PRPC from using the instances in the locked RuleSet versions that initially shipped with the application. If you add a rule instance to the work type and set its availability to Withdrawn, PRPC ignores the earlier instances of the rule in the work type when performing rule resolution.

Rule Resolution Process

Rule resolution works by assembling a cache of rules. If the correct rule instance is in the cache, rule resolution returns the appropriate instance and PRPC verifies that the user can actually process the instance. If the rule is not cached, PRPC must first add it to the cache. The following steps comprise the rule resolution process.

1. PRPC checks the rule cache to see if the rule is already in memory. If so, PRPC skips to step 8.
2. PRPC selects all rule instances matching the name (and type) requested.
3. PRPC discards rules with an availability of No/Draft Mode.
4. PRPC discards rules that are not in an applicable RuleSet version.
5. PRPC discards rules that are not in the class hierarchy.
6. PRPC discards rules with an availability of Withdrawn. The remaining rules are ranked by class (specific to most general), RuleSet version (highest to lowest), Circumstance (specific to most general), and Circumstance Date (today's date is past the cutoff date).
7. PRPC adds to the cache any rules that are left.
8. PRPC picks the best instance.
9. PRPC checks that the rule's availability is not set to Blocked.
10. PRPC checks that the user is authorized to use the rule.

Note: One caveat regarding RuleSet versions: rule resolution only looks for rules in a **single** major version. If an application is built on RuleSet version A:02-05-13, rule resolution will not look for rules in RuleSet version A:01-04-17.

Module 5: Creating Draft Flows and Draft UI

Lessons Covered:

- Introduction to Draft Flows
- Introduction to Draft UI
- Creating a UI Layout
- Repeating Layouts

Introduction to Draft Flows

Objectives

At the end of this lesson, you should be able to:

- Use draft flows to verify processes before creating property, UI, and business rules

Things to Know

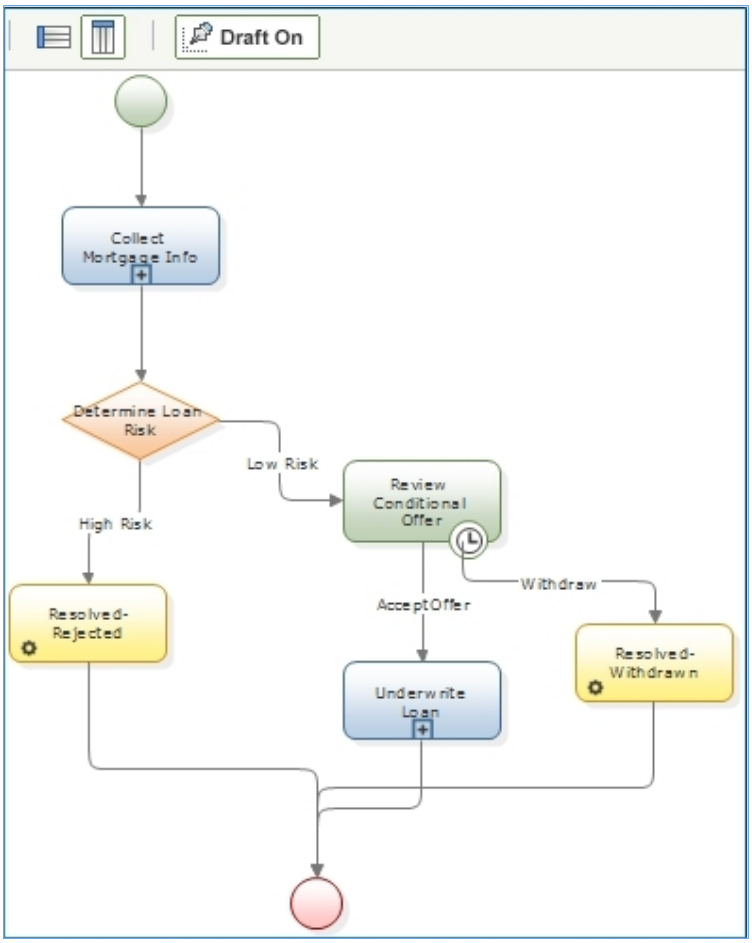
Draft Flows

Draft flows are a powerful feature in PRPC, allowing us to build and test a process without creating any of the rules that would otherwise be necessary. By using a draft flow, we can identify potential issues — which may affect the UI and/or decision evaluation — at a time when fixing them is fairly simple. We can create the underlying rules later, in the development phase of the project, after we've verified that the process logic is correct.

Draft flows are typically used during the DCO (Direct Capture of Objectives) process. When the initial process requirements are gathered, they are translated into a flow. This allows us to test and demonstrate the draft flow before beginning application development.

Creating a Draft Flow

When creating a draft flow, the important thing is to capture the *logic* of the process using flow shapes. Concentrate on the outcomes of decisions, the steps for human operators, and the integration points with third party systems. Don't focus on the rules; we will implement them as we progress through the development cycle. This allows us to create a framework for presenting a draft UI within the context of the business process and helps us to correct design issues before we begin developing our rules — properties, sections, decisions, etc.



Using **draft mode** allows us to save a draft flow. Normally, PRPC requires that we complete the required fields on the properties panel for a shape and that any specified rules exist, and prevents us from saving the flow if these checks fail. When we enable draft mode, PRPC still performs these checks, but allows us to save and run the flow. When it encounters a missing rule (such as a flow action), PRPC presents a basic UI listing the rule that would be run.

Draft Flow Design – Best Practices

Keep the following in mind when creating a draft flow:

- Create draft flows in the framework layer of work classes (if creating an application with a framework layer). When creating a specialized implementation, leverage the existing framework flow rules and copy them to the implementation layer.
- If a sequence of assignments are made to a single actor, consider implementing those assignments in a special type of subprocess called a screen flow.
- Associate use cases with flow shapes when possible.
- Complete the StatusWork and Instructions fields in the Properties panel for each assignment.
- Do not specify activities for a router shape, unless the activity already exists; merely focus on the *need* for a router at that point in the flow.
- Don't specify an activity on a utility or an integrator shape. The exceptions to this rule are fairly simple activities, such as **UpdateStatus**, that already exist.
- Add rule names to status and when connectors, to identify the purpose of the connector on the process diagram. Don't specify decision rules on decision shapes.
- Indicate the likelihood on each connector. This helps to verify the primary path.
- Name all flow actions.
- Use status values on the end shape of a subprocess to identify when processing passes back to the calling flow.
- Finally, remember to turn draft mode *off* when the flow is complete and ready for deployment.

Flow Reuse – Best Practices

One of the best practices for flow reuse is to create a **flow shell**. Whenever possible, keep the generic part of a business process in the framework layer. Move only the specialized portion(s) of the process to the implementation layer. We can run, or call, the generic flow in the framework layer from the implementation layer. Remember that the only shapes that *must* be present in a flow are the start and end shapes. All other flow shapes are optional, and their usage depends on our specific needs.

Remember that integration is typically a specialized flow function, and is rarely generic.

Reference Material

PRKB-25166 — How to create and test a flow model

Introduction to Draft UI

Objectives

At the end of this lesson, you will be able to:

- Explain what a draft user interface is
- Discuss the benefits of creating a draft UI
- Identify who should create a draft UI

Things to Know

What is a Draft User Interface (UI)?

During the initial stages of Direct Capture of Objectives (DCO), you gather user interface requirements. A draft UI is a set of PRPC rules that reflect these requirements. You can use controls to create mock-ups of application UI elements, even though the properties that will eventually hold user inputs and application outputs do not all exist yet.

During this stage, don't create properties. Instead, concentrate on how forms should look at each point in the workflows.

Everyone in the user community for your application, and everyone who is familiar with the business setting your application will support, has used many computer applications involving forms. A draft UI can provide all the stakeholders with a tangible, easy-to-understand (and easy to critique) preview of the scope and some details of the planned application.

Why Create a Draft UI?

Draft UIs allow you to share proposals with potential system users and other stakeholders, without requiring the extensive development effort that would be necessary to build a working, interactive user interface. Creating the draft UI isn't wasted effort, since the non-working, draft rules you create can typically be evolved and enhanced later to become a working, functional part of your application.

When creating a draft UI, focus on validating and completing the user interface capabilities needed to support each major aspect of the flows. At the same time, strive to gain stakeholders understanding and acceptance of the proposed UI.

Who Creates a Draft UI?

Generally, the business architect, in collaboration with user management, generates the draft UI during the Elaboration phase of DCO.

UI Rules to Use When Creating a Draft UI

Three primary UI rule types are involved creating a draft UI. These rule types are discussed further in the UI Rules lesson.

- **Harness** — Defines the top-level structure of a work item form (also called a user form).
- **Flow Action** — Presents the interface elements (forms to display, disposition options for work) to users to let them complete an assignment.
- **Section** — Contains UI elements (properties, labels, controls). Sections are the main visible building blocks of harnesses and flow actions. Sections can contain other sections.

Layouts are the compositional units of sections. A layout defines a table or grid that controls the placement of labels and input fields or output fields. On the Layout tab of the Section form, use the Layout menu button to add new sections, layouts, etc. Select a menu item, and drag it onto the section.

Best Practices

Concentrate on rapid UI mock-up during DCO sessions. Focus on sections and flow actions, and add sections to the corresponding flow actions. Whenever possible, choose the Applies To key and the RuleSet of the sections and flow actions you create so that they belong to the framework layer.

Be sure to complete the Short Description field on the rules you create, for clarity and understandability.

Use layouts and template controls to develop the appearance of the UI. Use headers to group UI elements such as fields and buttons, that are related.

Create a UI Layout

Objectives

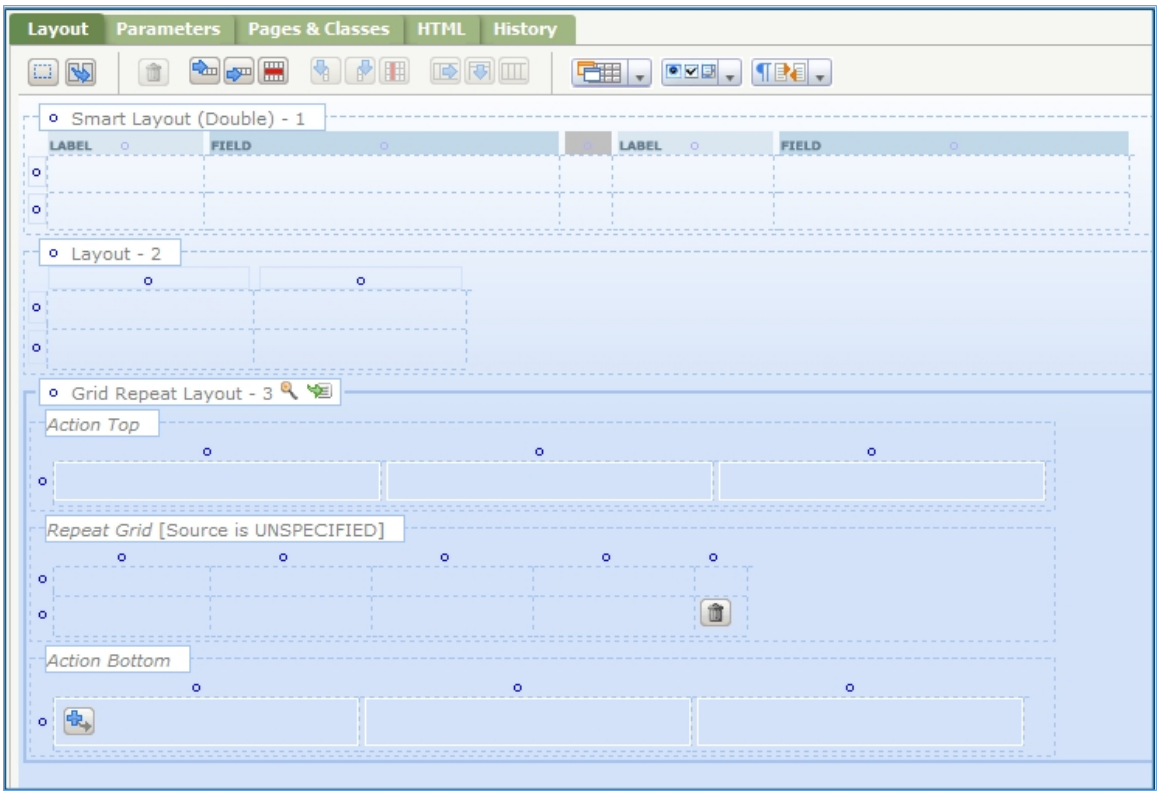
At the end of this lesson, you will be able to:

- Describe what a layout is
- Explain what layout types are available
- Format a layout

Things to Know

What is a Layout?

A layout defines a rectangular area on a form. Each section must contain at least one layout. You can reorder elements within the layout using drag and drop.



Layout Types

PRPC provides three types of layouts:

- **Smart Layout** — a grid of paired cells that has columns of uniform width, uniform styles, and uniform characteristics. Each pair of cells holds one label and one property value or other form control, and is separated by a spacer cell.
- **Free Form Layout** — an unformatted grid with the specified number of rows and columns, which do not follow the "label-field-spacer" order.

- **Repeating Layout** — an unformatted grid with an arbitrary number of rows, which enables users to perform complex editing on large lists of items. This type of layout is explored in more depth in the *Repeating Layouts* lesson.

Smart Layouts

A Smart Layout enforces vertical alignment, thus enabling us to create forms that have a consistent look and feel across the entire application, with less effort. This vertical alignment hold even when layouts are nested. When a section containing a Smart Layout is nested within another section, all labels and fields are given consistent widths. PRPC adjusts padding and cell widths to seek attractive spacing and presentation. Pega recommends you use Smart Layouts for new applications.

Free Form Layouts

Free Form layouts provide a flexible design, which makes them well-suited for presenting nested sections, controls and so on. Free Form layouts can be converted from a Smart Layout by selecting **Allow Changes to Columns** in the Layout Properties panel. Columns can then be added or deleted and the column type can be modified by clicking on the column and editing the properties panel. However, once you change a Smart Layout to Free Form, it cannot be changed back.

When to Use a Smart Layout or a Free Form Layout

Smart Layouts provide a consistent look and feel across the entire application, while Free Form layouts offer flexible design. Use the following table as a guide to determine which layout to select for a UI layout.

Smart Layout	Free Form Layout
Intended for displaying fields and labels.	Intended for presenting nested sections and large controls.
Better for presenting data in tabular form.	Better for positioning a UI element in a specific location.
Reliance on templates enforces consistent positioning of fields and labels from layout to layout.	Cells can be resized and merged to better accommodate extremely large or small fields.

Formatting a Layout

Use the Properties panel to control the appearance of a layout.

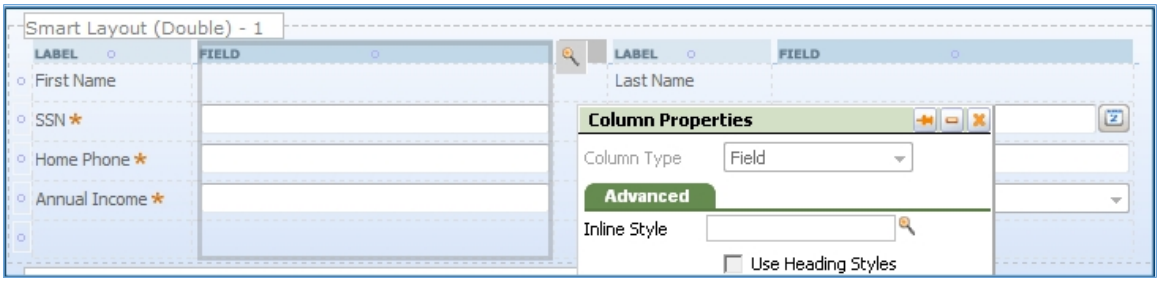
Headers

We can apply a Header or a Sub Header style to a layout or a section embedded in a section. This primarily affects styling and appearance, which can be reviewed and modified using the Branding wizard. Select No Format when you want the style to be dependent on the containing layout. There are four main types of headers:

- **Bar** — Used to show a header title. Cannot be minimized or collapsed.
- **Collapsible** — Used to make a layout collapsible. Expanded by default.
- **Accordion** — Used to stack several collapsible layouts on top of each other.
- **Tab** — Used to display layout titles side by side.

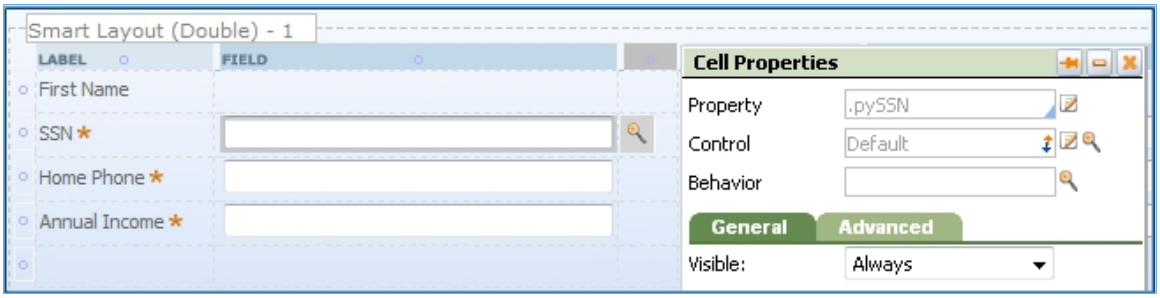
Row/Column Attributes

Customize a column or row by selecting its header.



Field Attributes

The Properties panel allows cell characteristics to be set, including maximum and minimum sizes, fonts, color, etc. Some attributes are only available when using a custom template. Set the cell's Control type from here.



Referencing One Section Within Another

We use sections within sections when we want to reuse a section or position a section within a layout. The Properties panel indicates that the cell contains a section.

Repeating Layouts

Objectives

At the end of this lesson, you will be able to:

- Describe the differences between the Grid, Tree, and Tree Grid Layouts
- Use the Grid, Tree, and Tree Grid Layouts
- Explain how to add different repeating layouts to a section
- Use the Layout Features
- Describe and effectively use Edit Mode

Things to Know

Both a Smart Layout and a Free Form layout are well-suited to display a fixed, predetermined amount of data, such as a mailing address. If we want to present a list of items of arbitrary length, what are our options?

We can **use a repeating layout to display an arbitrary (1-n) number of data items**. A repeating layout consists of one header row that contains property labels, and a row of property fields. When users encounter a repeating layout, they can add or delete rows in the layout as they need. Typically, repeating layouts are used to display properties in page list or value list properties.

PRPC provides three types of repeating layouts which extend the existing repeating layout functionality:

- Tree
- Grid
- Tree Grid

These layouts also provide multiple edit modes:

- Read Only
- Read / Write
- Modal Dialog
- Inline
- Embed
- Expand Pane

Adding Repeating Layouts to a Section

PRPC provides us three ways to add a repeating layout to a section:

- **Drag a page list or value list property** from the Application Explorer or Rules Explorer onto a section. When we release the mouse button, a dialog appears asking us to specify which repeating layout to use (Grid, Tree, or Tree Grid).
- **Drag a report definition** from the Application Explorer or Rules Explorer onto a section. When we release the mouse button, PRPC adds either a Grid layout (for list-type reports) or a Tree Grid layout (summary-type reports) to section. The layout can only reference properties listed in the report definition.
- **Drag a Layout from the Layout palette** onto the section. In the dialog that appears, select the Repeating radio button and the type of repeating layout you want to use (Grid, Tree, or Tree Grid).

Grid Layouts

Grid layouts display values from a single page of data. This type of layout allows users to view (and possibly edit) values of a Page List or Value List property or a Report Definition in a spreadsheet format when large amounts of data need to be displayed. This is an improved alternative to Row Repeat, Column Repeat, or Tabbed Repeat controls. The developer specifies which page property to display in each column, and each row corresponds to a single embedded page.

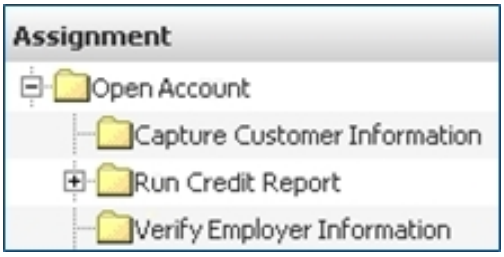
The grid is created dynamically, and has as many rows as is needed to display the requested data. It may be read-only, simply displaying property values, or editable. You may specify that the data can be edited in a number of ways:

- All the editable fields may be displayed as editable, as in a spreadsheet
- Editable fields in a row become editable when a user clicks on that row
- A modal dialog appears with the editable fields for a row when the user clicks on that row

Depending on parameters, users may be able to add or delete rows, to reorder rows, and to apply actions to rows by clicking action buttons. The action buttons may appear in areas above or below the data. Express action buttons can appear in each row, or only in the row that users click.

Tree Layouts

Tree layouts display objects with a nesting relationship, such as covers and work items. This type of layout allows users to view the embedded pages of a Page List, or the entries in a Value List property, as a tree. Users can click branches to display or hide their leaves. The tree can display embedded relationships as deep as twenty levels. The developer specifies which page property to display as the name of each branch; each branch corresponds to a single embedded page or property.



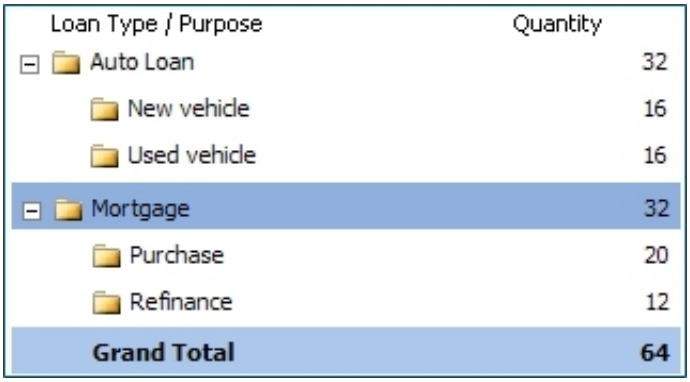
Tree Grid Layouts

Tree Grid layouts combine the nesting relationship capabilities of a tree layout and the value display capability of the grid layout. This type of layout allows users to view (and possibly edit) values of a Page List or Value List property in a display that combines the navigation benefits of a tree with a spreadsheet's access to data, as an improved alternative to Row Repeat, Column Repeat, or Tabbed Repeat controls when large amounts of data need to be displayed.

Developers specify which page property to display in each column, and each row corresponds to a single embedded page.

The tree grid is created dynamically, and has as many rows as is needed to display the requested data. It may be read-only, simply displaying property values, or editable. Developers may specify that the data can be edited in a number of ways:

- All the editable fields may be displayed as editable, as in a spreadsheet
- Editable fields in a row become editable when users click on that row
- A modal dialog appears with the editable fields for a row when users click on that row



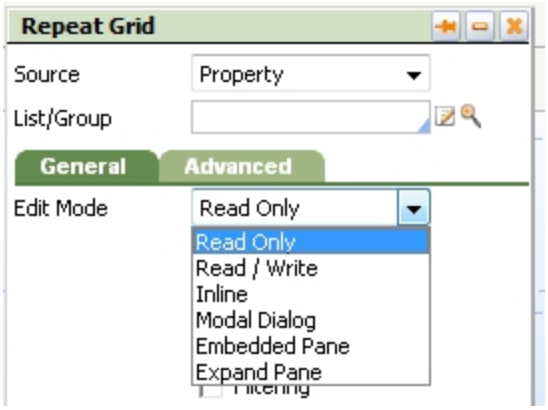
Layout Features

We can add non-repeating elements to a repeating layout. For example, we can add buttons linked to actions such as Add or Refresh, or add descriptive text or a header/footer. Repeating layouts allow users to drag and drop items to reorder the list. This feature also works with row/column repeating layouts.

The class of the page/page list appears automatically on the repeating element within the section.

Edit Modes

Four edit modes are available to use for repeating layouts:



- **Read Only** — Data displays as non-editable.
- **Read / Write** — Data that can be edited is editable.
- **Modal Dialog** — Data displays as non-editable. Double-click the row to open a modal dialog for editing data.
- **Inline** — Data displays as non-editable. Double-click the row to edit data in place. With this mode, the layout updates without refreshing.

Module 6: Creating a Data Model

Lessons Covered:

- Properties
- Define Properties Wizard
- Using a Data Model
- Data Classes
- Data Tables
- The Clipboard

Properties

Objectives

At the end of this lesson, you should be able to:

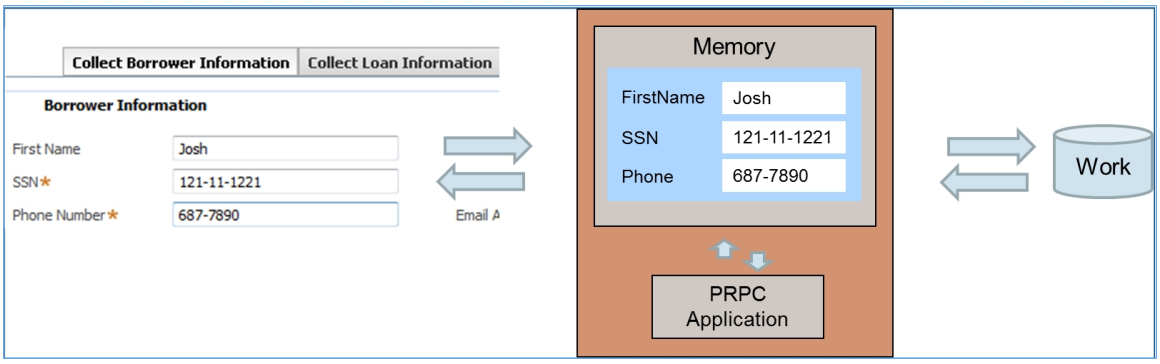
- Describe how properties are used in an application
- Locate and use properties
- Create simple properties in an application
- Examine the properties in an application
- Format properties for display with controls

Things to Know

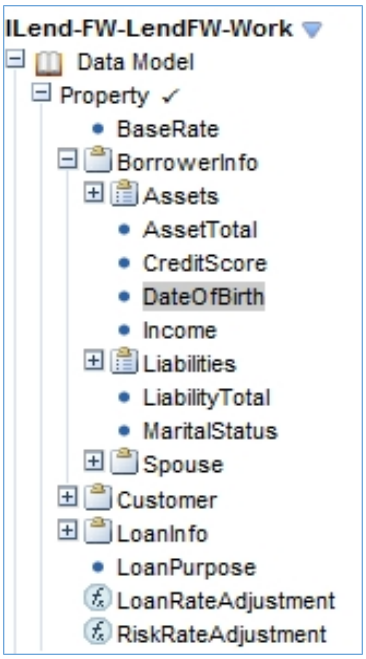
PRPC Properties

All PRPC applications capture, store, process, and display data that is saved as properties. Many property values are set through user input on an HTML form, such as a customer's name or their billing address. Other property values may be hidden from view and set by a computation, such as calculations, confidential information, or an index value for a table.

The properties used for a persistent object are stored in the database. A user or PRPC application creates a work item, retrieves it from the database and brings it into memory to update the data (most clipboard pages contain values associated with a running process and its work items). The work item's properties correspond to "fields", or in many situations, to relational database columns.



Properties are found in the *Data Model* category in the Application Explorer.



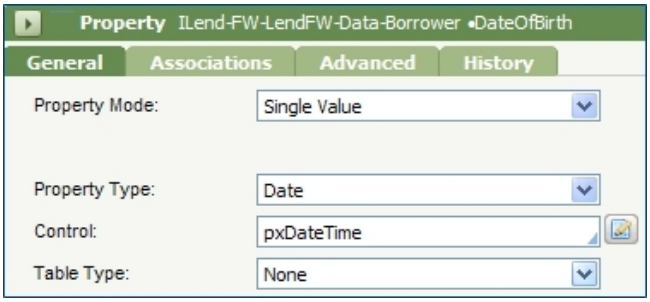
Property Names - Best Practices

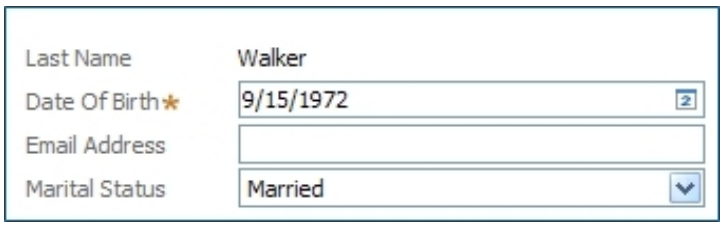
The naming following conventions promote consistency and ease-of-use:

- Use whole words, instead of an abbreviation, acronym, or a slang expression.
- Use names that are intuitive or familiar to business application users. Avoid using technical jargon.
- Use only alphanumeric characters and dash characters; do not use special characters.
- Start every property name with a letter.
- Use CamelCase for property names, capitalizing the first letter of each word in the name, such as UseShippingAddress.
- Remember that property names are case sensitive; "emailaddress" and "EmailAddress" are two distinct properties.
- Do not use @ and $ in a property name even though they are legal Java identifier characters, as they are not legal characters in a property name.
- Do not use punctuation marks in a property name, for example, dashes, dots, etc.
- Do not exceed the 64 character limit. However, if the property is of mode Single Value and may be useful for searching, the name can be longer, but should not exceed the maximum column name length allowed by the software supporting the PegaRULES database.
- **Do not create a property using a name that starts with px, py , or pz**. These prefixes are reserved and identify standard properties, and should **only** be used when overriding a standard property in an application.
- **Use distinct, unique property names** within an inheritance path to avoid poor runtime performance. Instead of using a property named Amount, use more descriptive names such as PrincipalAmount or TotalOrderAmount. When creating a property using the same name as an existing property in the inheritance path, PRPC presents a warning message and confirms that want to create the property.
- Don't use a property name that matches a reserved page name or keyword, such as Top, Parent, Local, Param, or Primary.

Property Types

The property type determines the type of data that the property will contain (the format and characters allowed for the value), the display of the associated fields and what users can do with them.





A property type may be scalar (consisting of a single text string or a single page) or aggregate (containing multiple, individually named values).

Single Value Property Types

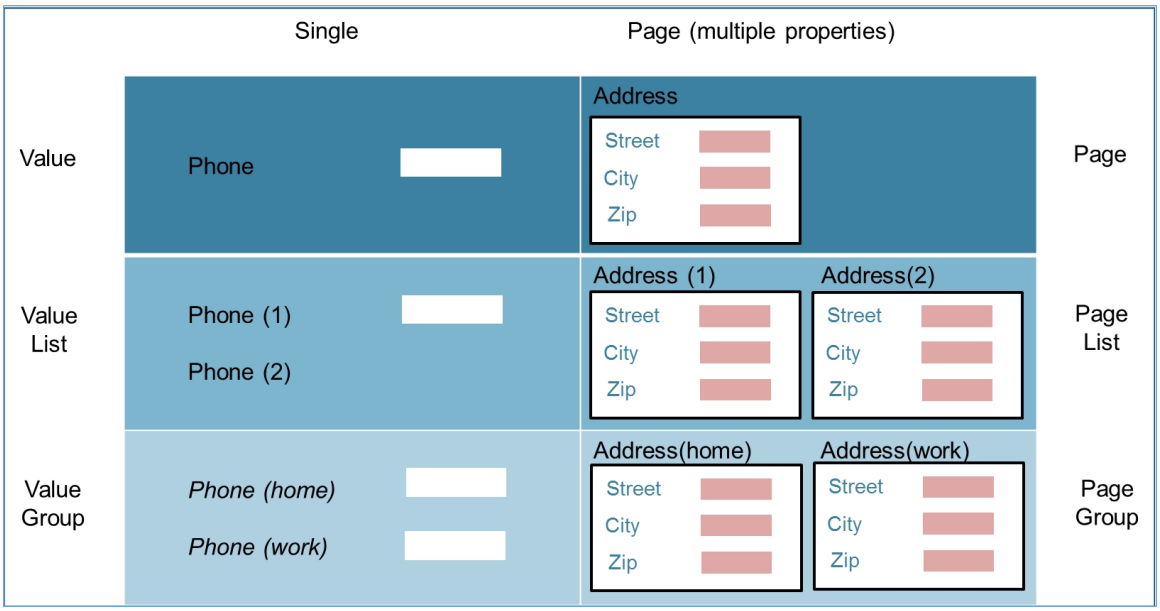
Type	Description
Date	Date with no time
DateTime	Date and time of day
Decimal	Fixed decimal number; positive, negative, or zero
Double	64-bit floating point number; positive, negative, or zero
Identifier	Alphanumeric string that can be a Java identifier, with no line breaks
Integer	Integer; positive, negative, or zero
Password	Text string that ordinarily is not displayed when typed into a form
Text	Unedited text, which may contain spaces, tabs, and line break characters
Time of Day	A time of day, in the form HH:MM:SS
True or False	Boolean (two values)

Aggregate Property Modes

Aggregate properties provide facilities similar to arrays, repeating groups, and unordered sets or collections found in other development tools.

By default, every value that we create is a single value property. We can change the mode of the property to any of the following modes:

- **Value List** — A single property that has an ordered list of none, one, or many sequentially numbered strings as values, indexed by a numeric index (subscript) starting at 1.
- **Value Group** — A single property that has an unordered set of values of any Type. It may have none, one, or many values as values, each uniquely identified by a constant string value as subscript.
- **Page** — A data structure that holds name-value pairs which may be contained in memory, or stored in the database. The system has many types of pages — named pages, unnamed pages, embedded pages, parameter pages, and so on. A page may have an associated name, a class, and a list of unique names (with values for each of these names), and messages. In most cases, the names identify properties defined in the class of the page. Values may be text, or may themselves be a page or multiple pages, extending the page data structure.
- **Page List** — A data structure consisting of an ordered list of zero or more pages, but the order of the pages is not significant. Pages are identified by an integer index (starting with 1) and have sequential subscripts.
- **Page Group** — A data structure may be empty or can contain one or more pages, but the order of the pages is not significant. Each page is identified by a string subscript value.



There are also five Java properties that are used to implement the data model for a Java class of external Java objects.

We'll cover property mode in greater detail in the *Using a Data Model* lesson.

Creating a Property

When creating a property, we need to provide the following information:

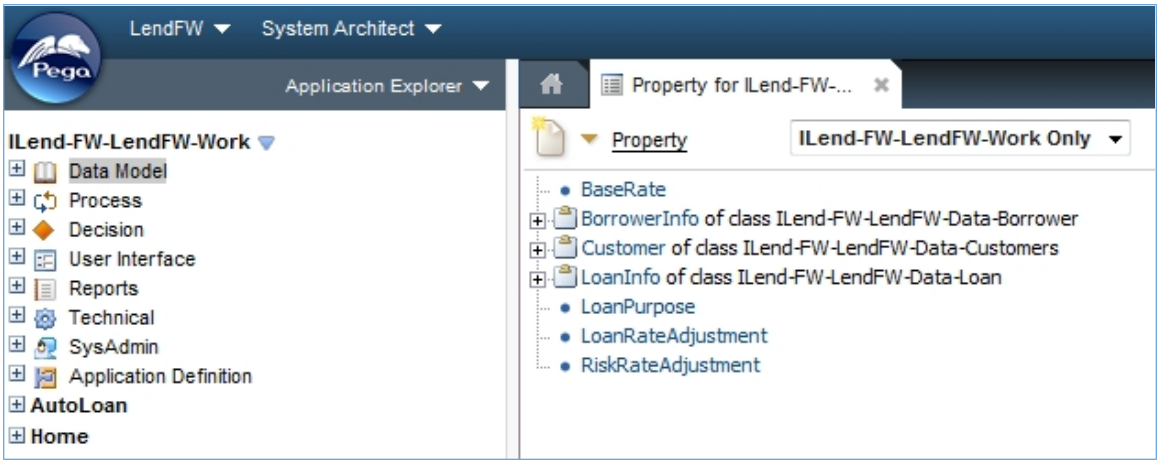
- The **name** of the property.
- The **class** to which we want to apply the property.
- The **RuleSet version** in which we want to store the property.

If we're creating a single-value property, we also need to provide the **type** of the property. We can either click **Quick Create** to create a property with this information and skip the property rule form, or we can click **Create** to continue to the property rule form and enter additional information such as display formatting and default values.

If we're creating a page mode property, we ignore the property **type** and must click **Create**. On the property rule form, we must specify the **property mode** and the **data class** to which we want to map the property.

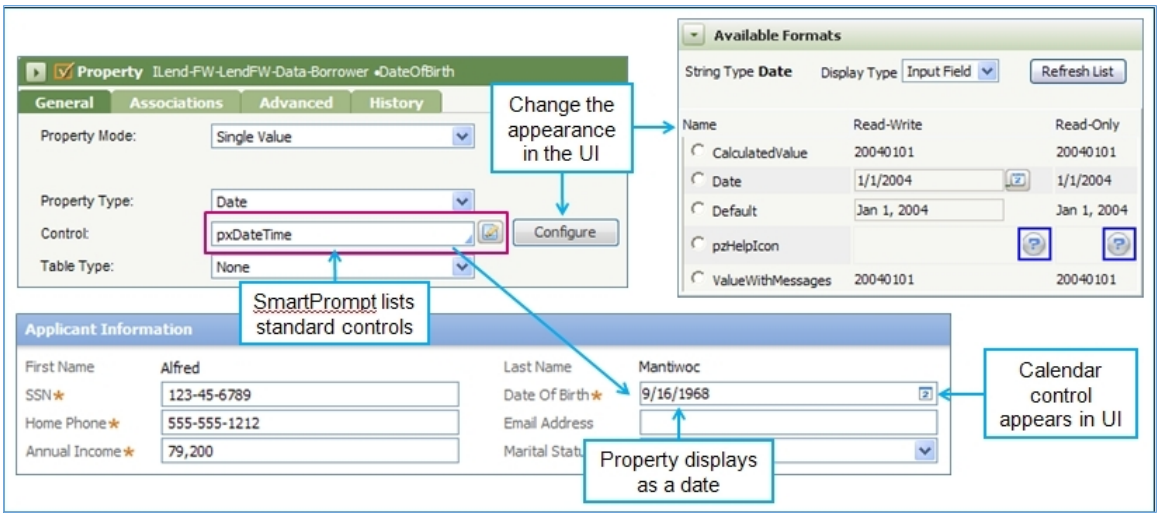
Examining Properties in an Application

To examine the properties in your application, click Data Model.



Controlling Format and Display with Controls

The General tab of the property rule form includes a Control attribute that controls the display of a property. For example, the pxDateTime control adds a calendar control to the user interface and displays the property either as a date or a date and time.



Define Properties Wizard

Objectives

At the end of this lesson, you will be able to:

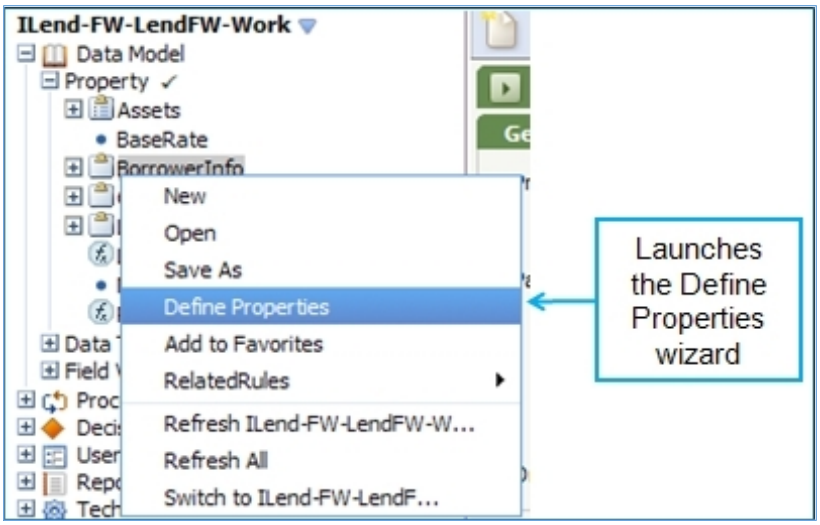
- Use the Define Properties wizard to define a property

Things to Know

Using the Define Properties Wizard

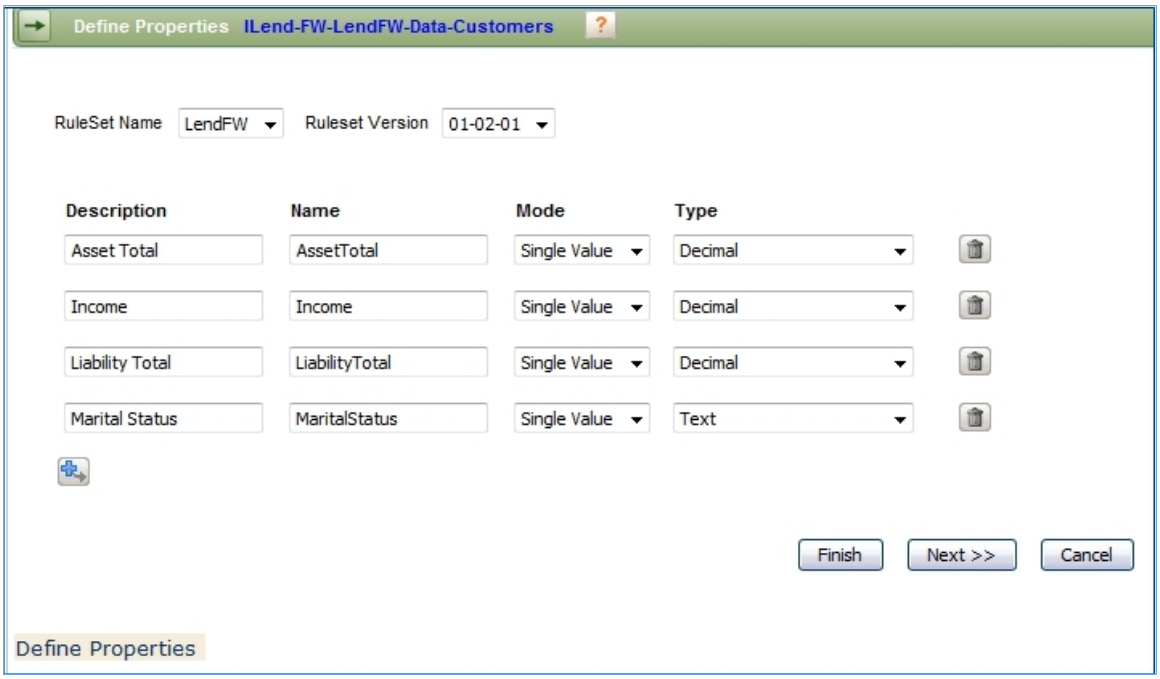
The Define Properties wizard is a tool used to create one or more properties with a common Applies To class at once, by completing a single form. When you need to create several properties, using the wizard is faster than completing the New Rule dialog for each property.

This tool is especially useful when creating the set of properties that should be grouped in a page at runtime, for example, for an Address data class that has a collection of single value properties.



Define a Property

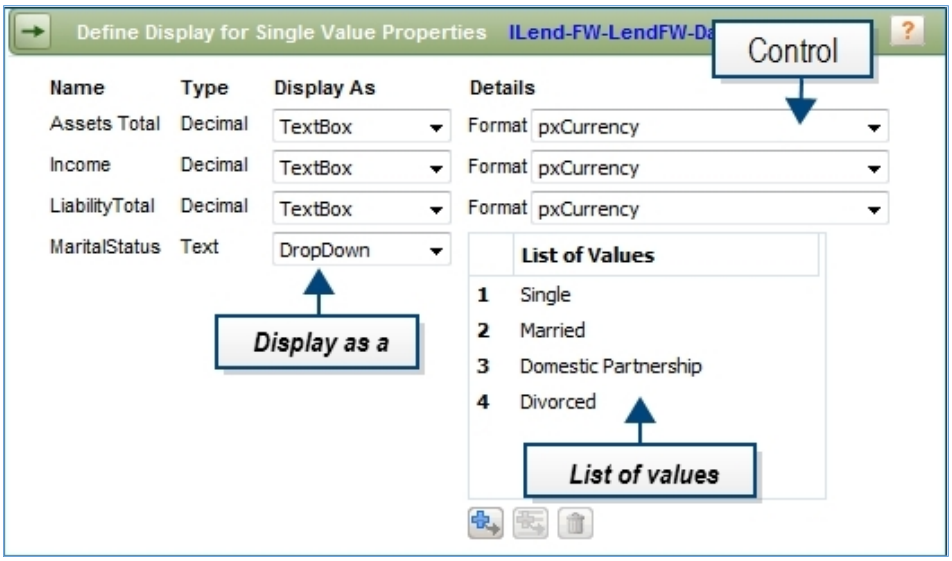
Complete the Define Properties wizard to capture property information such as description, name, mode, and type for each property. The wizard uses this information to create the necessary property rules.



Define the Display of a Property

On the second step of the wizard, we can specify the display and formatting options for each property. Our choices depend on the property type on the previous step of the wizard.

For example, if we want to display a text property as a drop-down list, we can enter a list of the possible literal constant values that the property may have.



Using a Data Model

Objectives

At the end of this lesson, you will be able to:

- Describe and construct a data model

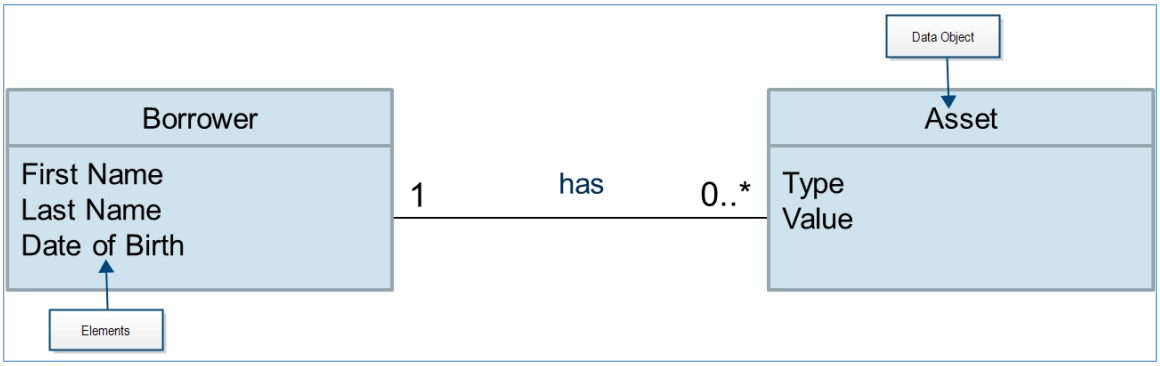
Things to Know

What is a Data Model?

A data model describes data entities that the business needs to record, for example, customer information, applicant information, and other aggregate properties. It describes the organization and structure of these data objects, data object elements, and their associations.

The data model also provides information about the nature or characteristics of relationships among the data entities such as ownership, parent-child, one-to-one, one-to-many. Below are several examples:

- A Borrower can own one or more assets; a borrower can request only one loan.
- An Asset data object has Type and Value data elements.
- An Asset is associated with a Borrower, which has other data elements as well.



When you create a property, you must specify its mode and type. For property mode you can specify one of eleven property modes, including:

- String-based modes
- Page-based modes
- Modes used with Java objects

The property type determines the kind of data that values of the property represent, for example Text, Integer, DateTime, or Password, and can affect the format and allowable characters in the value.

Usually the data model for an application is created by the Lead System Architect on the project, but the System Architect must understand how to interpret and how to implement the data model.

How to Create and Map a Data Model

There are four main steps to creating and mapping a data model:

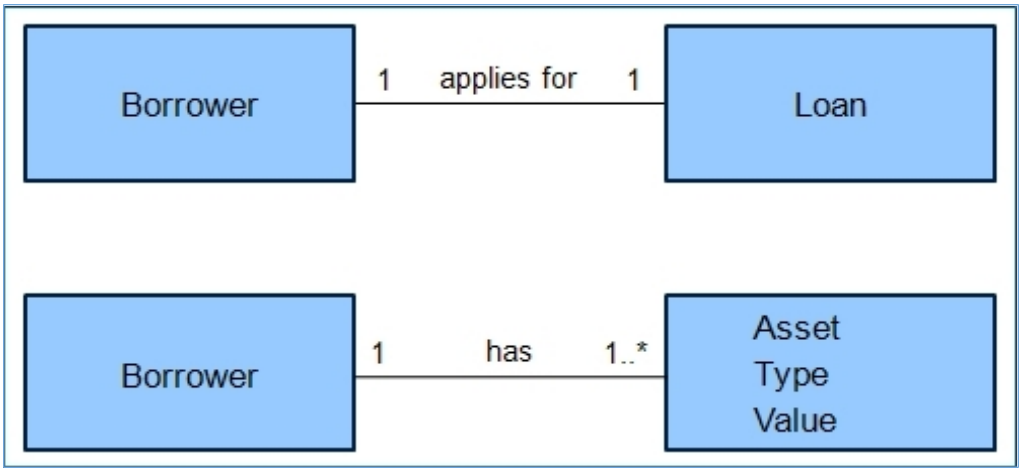
- Identify and Associate Objects
- Consider General and Specialized Objects and Elements
- Map Objects and Elements
- Add Properties to Data Classes.

Identify and Associate Objects

Identify key data objects (illustrated by the top diagram) and compound elements (bottom diagram). Consider how they are associated with other objects and the cardinality of the association.

Cardinalities specify a range, expressed in the form min..max, where an asterisk means "many".

For example, 0..1, 1..1, 1..*, etc.



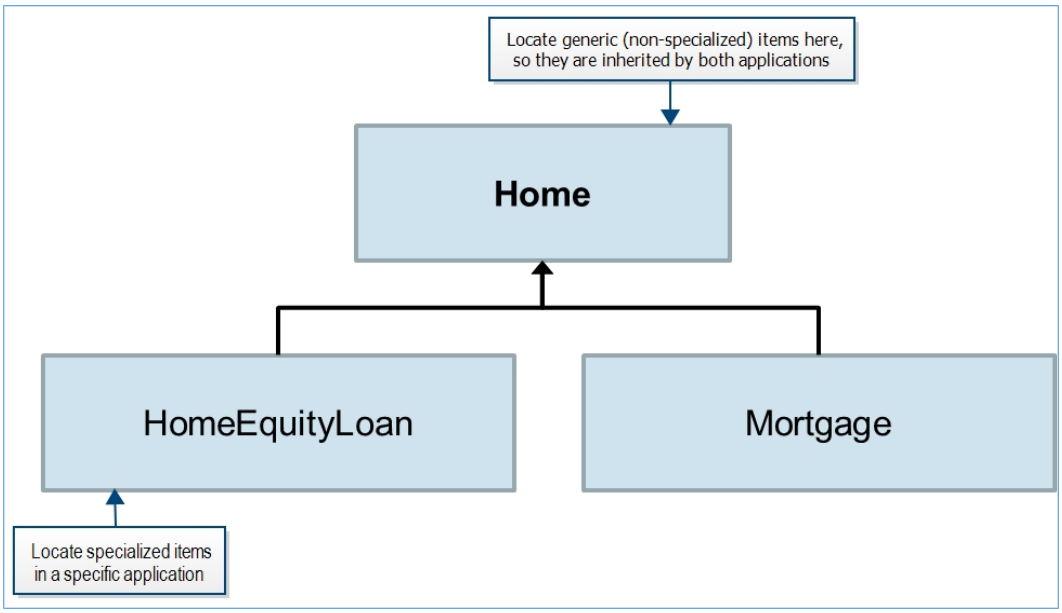
Consider Generic and Specialized Objects

Identify which data objects are generic and which are application-specific. Specialized elements are the ones that reuse the properties and rules of generic elements. Always think about reusability.

For example, consider the following two properties:

- HomeType — describes the type of building, such as single family or multi-family home, or condominium.
- Terms — describes the terms of the loan such as 5 year, fixed rate.

We can define the HomeType property under Home, because it can be used by both the Home Equity and the Mortgage applications. The Terms property, which may include default values specific to each loan type, should be defined for each work type.



Map Objects and Elements

You can translate the data model to PRPC classes and properties.

Data objects correspond to data classes. Data classes inherit from other data classes, allowing reuse, while providing unique specialization. All data classes are derived from the class Data-. For example, the class BorrowerInfo inherits from the class Data-Party. BorrowerInfo accesses all rules defined in its parent class, but also includes specialized data - in this case, borrower information.

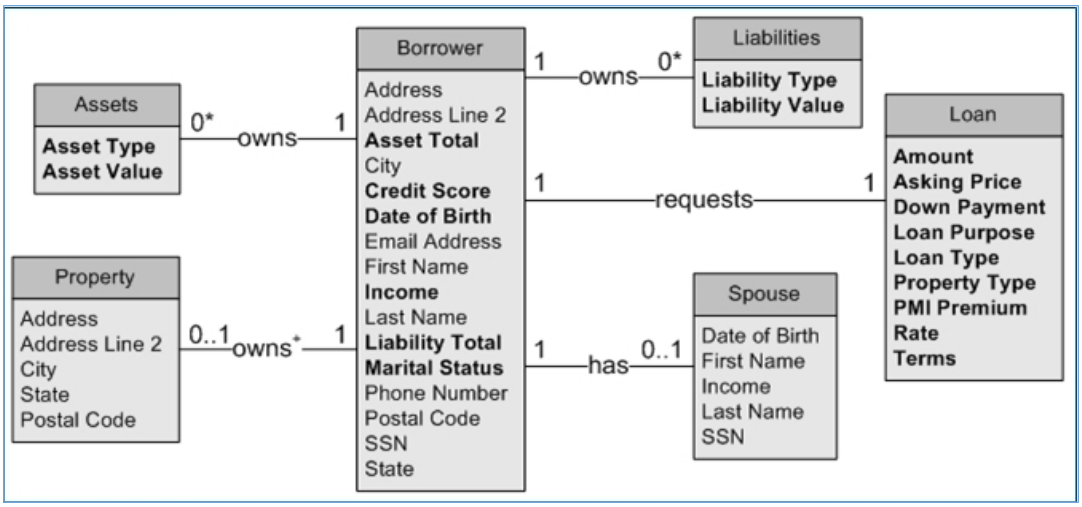
Elements correspond to properties, which are applied to data classes. Single-value properties contain simple text, integer values, dates, Boolean values and amounts. For example, the property named Work-.pyID holds the system-assigned unique identifier for a work item.

A compound element, such as a property of mode Page, Page List, or Page Group, is composed of multiple properties. When specifying the name of a compound element, use the plural form if more than one object can be associated. Below are several examples:

- LoanInfo (single structure) is an appropriate name for the case where the borrower requests one loan.
- Assets (plural structures) is an appropriate name for the case where the borrower has multiple assets.

Add Properties to Data Classes

The first step is to identify all of the properties that pertain to each object. Next, we create the properties, using a consistent naming style for all property names, and specifying the mode and type for each property. Start the property name with a letter and use only letters and digits (no spaces). For mode, specify Single Value, Page, Page List, etc.. For type, specify Text, Integer, Date, (if the mode is Single Value); or a class name (if the mode is Page).



Data Classes

Objectives

At the end of this lesson, you should be able to:

- Implement and extend a data model
- Add properties to the proper data class
- Create page properties to reference the data model

Things to Know

What is a Data Class?

Once the data model for our application has been diagrammed, reviewed, and approved, we can implement it in PRPC. **Each distinct entity in the data model corresponds to a data class**, and the characteristics or elements of that class correspond to its properties. The data class allows the properties we define in the data model to be used by our application. To use a data class in an application, we must create a property in the work class to reference the data class. This is often called an aggregate property and is implemented using one of the following three modes:

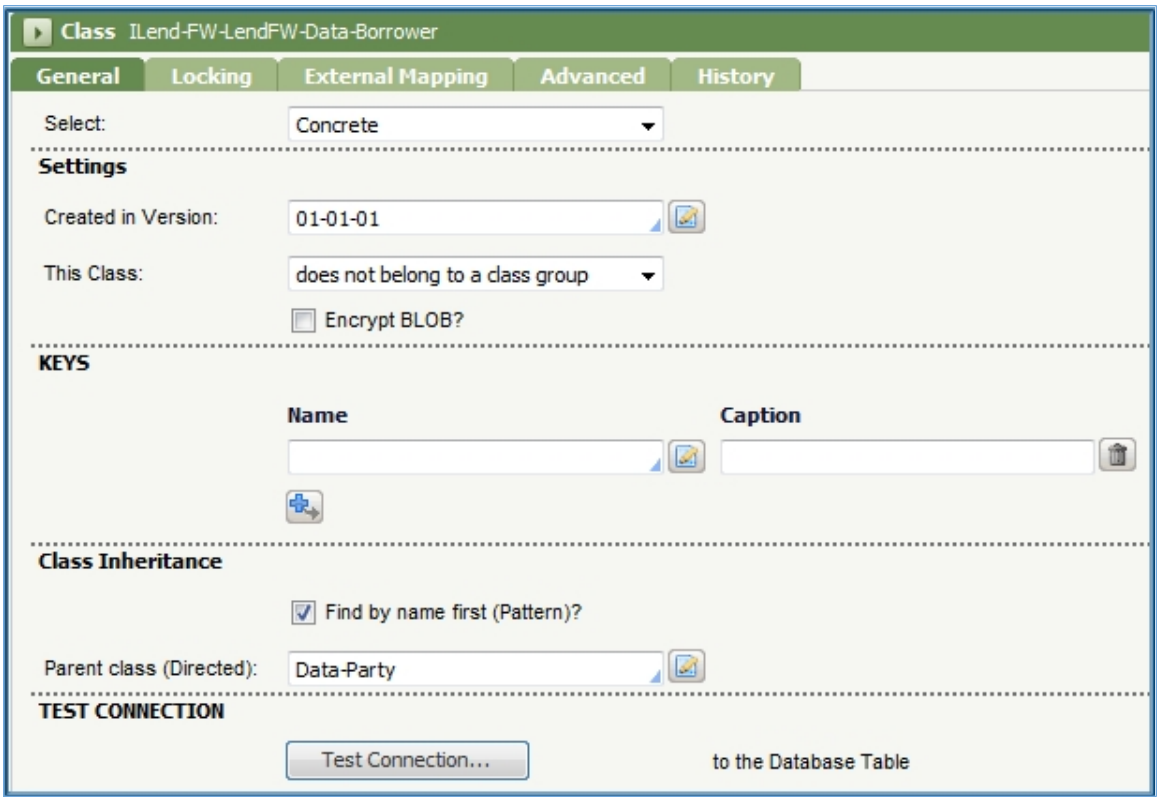
- **Page** — Used to collect data about one entity.
- **Page group** — Used to collect data about a small number of entities, where each entity has a syntactic identifier and where order isn't important, for example, home address, work address, and billing address.
- **Page list** — Used to collect data about a number of entities with no upper bound, where each entity has an ordered identifier assigned by PRPC.

The properties in the data class become elements in a page, page group, or page list, and are displayed in the Application Explorer as in the following example.

The data that we want to collect should be part of the data model. Data classes should contain instances that are persistent. By contrast, properties that will be used to process the work item, but that don't need to be recorded in a separate system of record, should be defined in the work class. (All properties in a work item are persisted, regardless of their class. Data classes are used for logical groupings of objects and to be able to write to and query against tables other than the work item table.) Like work classes, data classes can inherit properties from other data classes, such as Data-Party. This allows us to reuse existing properties rather than create new ones.

Creating a Data Class

When creating the data classes needed for your application, you must create a class rule and specify the directed parent for the class to use the existing properties in that class.



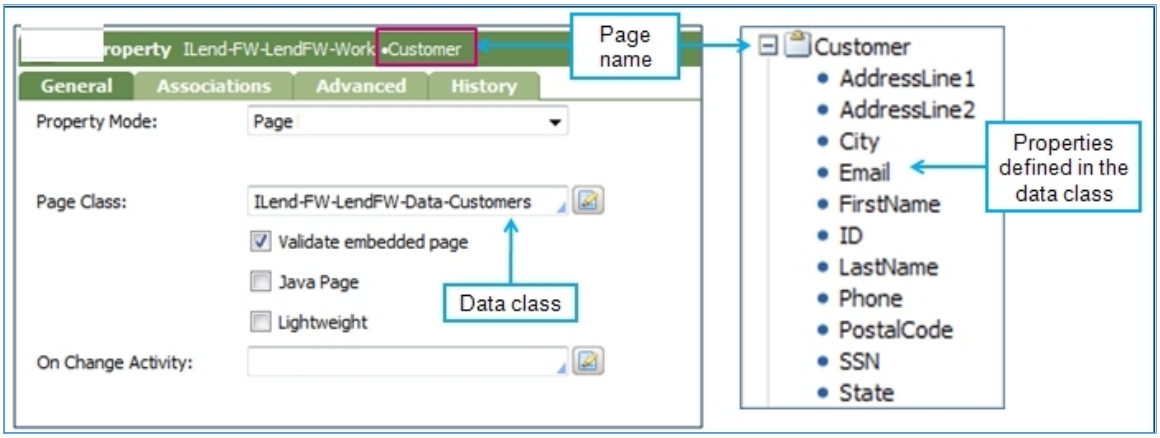
Adding Properties to a Data Class

Before you begin adding properties to a data class, we recommend that you identify all of the properties that pertain to each object. It is a best practice to use a consistent naming style to create each property.

Specify the appropriate property mode, such as Single Value, Page, or Page List, and the property type. For example identify, Text, Integer, Date, (Single Value); Page Class (Page, Page List, Page Group).

Creating Page Properties

To create page properties for an embedded page, select one of the three property modes — Page, Page List, and Page Group — that identify properties for which the value has a single or multiple page structure.



Referencing Data Classes and Properties

Pages can contain other pages (known as embedded pages) to an arbitrary depth.

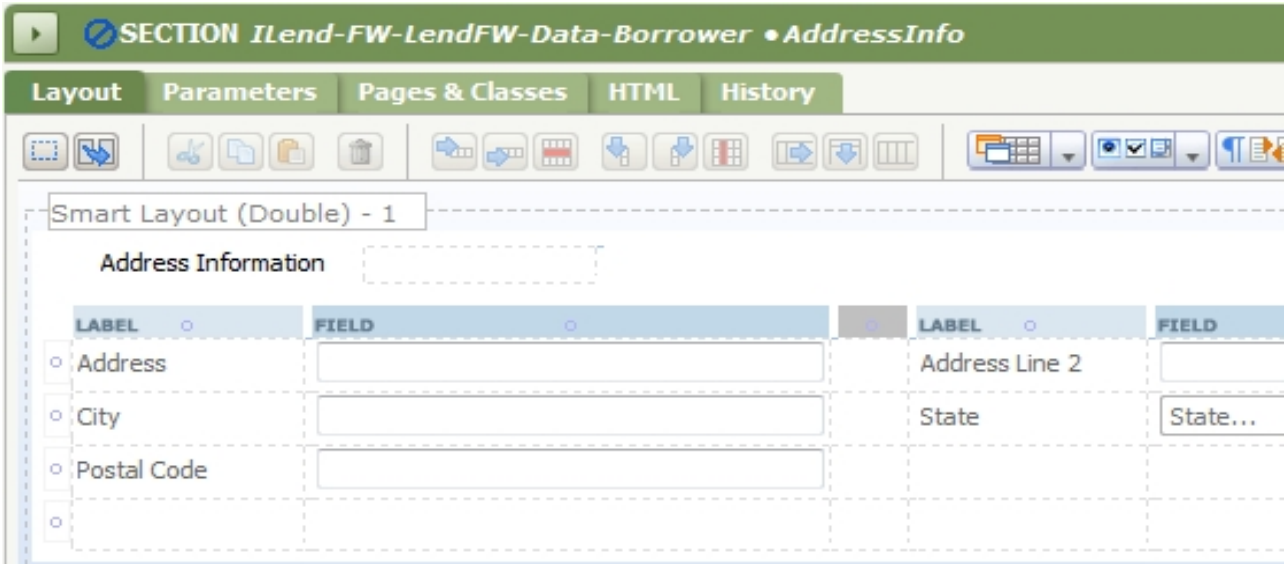
- Reference a single value property as .PropertyName.
- Reference an embedded property as .PropertyName.PropertyName, or PageName1.PropertyName.PropertyName.

A property of mode page has an associated page class. For example, the Customer page property has a page class of ABC-FW-SalesOrderFW-Data-Customer. This property, like all properties, belongs to a top level page. In most cases, it will be pyWorkPage. That is why in the clipboard we will see this as a page inside the pyWorkPage and why it's called an embedded page. The fully qualified reference to an embedded property begins with a reference to its top level page. For example, to reference the .DateOfBirth property on the BorrowerInfo page, use BorrowerInfo.DateOfBirth.

Best Practices

When possible, create rules and the properties they reference in the same class in order to enforce integrity. Derived classes will inherit these supporting rules. This facilitates reuse and provides greater consistency throughout the application. It also automatically establishes the proper context for the properties.

For example, we should create the section, shown below, in the same class as the properties that are referenced in this section (Address Line 1, City, State) or in a class that can inherit these properties.



Data Tables

Objectives

At the end of this lesson, you will be able to:

- Describe a data table
- Create and modify a data table
- Create data table instances

Things to Know

A data table is a simple structure that is usually used to store reference data. This data is stored outside of the application, in a separate data table. This allows us to update the information in the table without modifying the application. A data table contains instances that are commonly used to populate dynamic list controls. For example, a data table might contain currency codes and currency descriptions, such as USD for United States Dollar or GBP for Great British Pound.

Informally, a data table is PRPC's internal representation of a "flat file" that contains no repeating groups; it contains only scalar (Single Value) or Value List mode properties. A data table consists of the rows of a concrete class, often derived from the Data- base class, and those that do not require user input for properties inherited from any higher classes.

Note: Do not create a data table for a class derived from Work- classes because work items are not designed to define reference data, although they may contain properties that have values obtained from data tables.

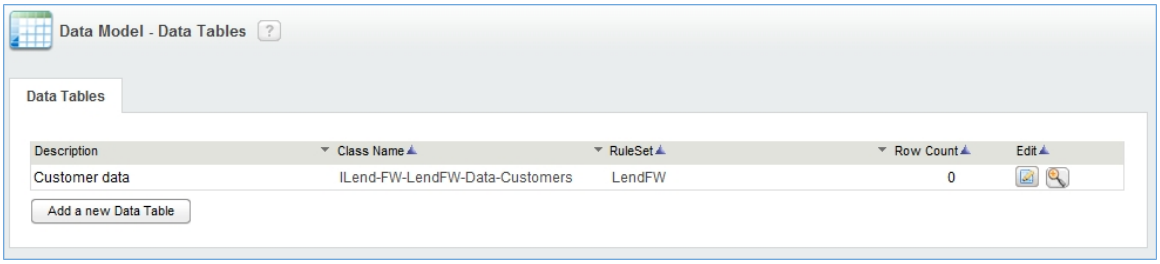
The Data Table Wizard

The Data Table wizard allows users to interactively enter and update data instances for classes that have a simple structure. The wizard creates:

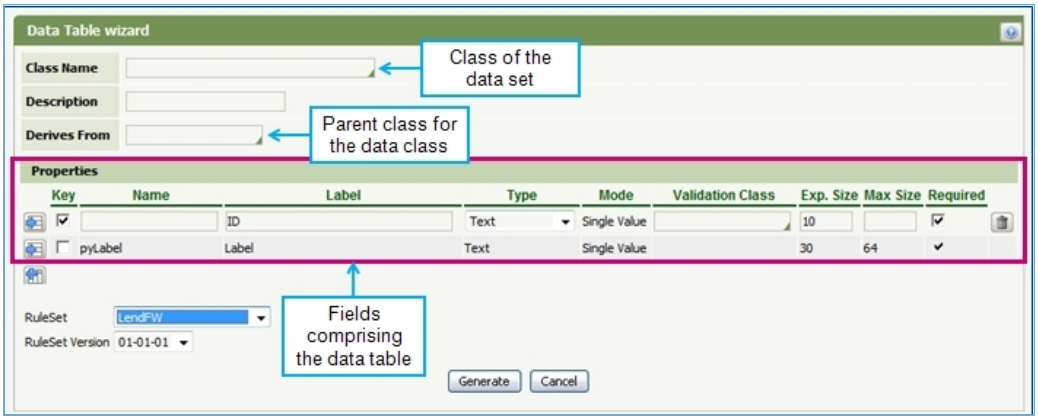
- A concrete data class.
- The properties in the data class.
- List rules, editing rules, and validation rules.
- A data transform rule to supply initial values for properties.
- Optionally, a dedicated table in the PRPC database and a corresponding database table data instance.

Note: By default, the Data Table wizard maps the classes it creates to the *pr_other* table in the PRPC database. Data tables that either contain hundreds of rows of data, may be updated frequently, or may be moved to other PRPC systems should use a dedicated table, rather than *pr_other*.

To create a data table, click **Data Model>Data Tables** from the Pega menu to access the Data Tables landing page. On the landing page, click **Add a new Data Table** to launch the Data Table wizard.

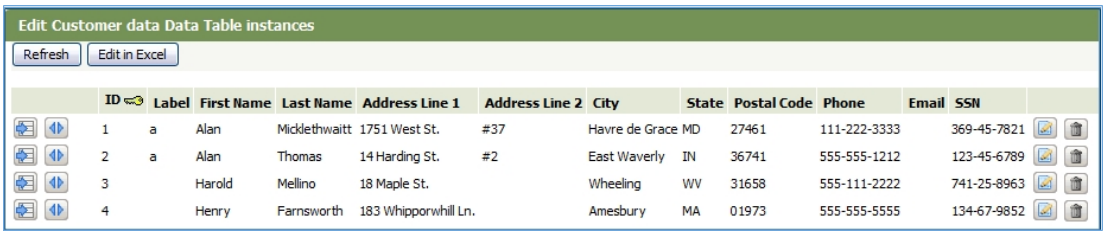


Data Table Wizard Form



Data Table Editor

Ordinarily, the Data Table editor is used to work with the instances of data classes that were created by the Data Table wizard (these classes are also called data tables). Specifically, we use the Data Table editor to add, update, or delete instances of a concrete class.



The Data Table editor enables us to maintain instances of the class in the data table. Microsoft Excel can also be used to add rows, update rows, and delete rows, although we cannot maintain Value List properties with Microsoft Excel.

Data Table Instances

Data table instances define each data instance (table row), that can be read by an application. Data table instances are created to associate an existing class or class group (and other classes derived by pattern inheritance from that class) with an existing relational database table or view in the PegaRULES database or an external database. Data table instances can also revise existing class-to-table relationships. Use with large tables to query for a specific set of rows based on their key or label values and edit them.

The Clipboard

Objectives

At the end of this lesson, you will be able to:

- Describe what the Clipboard is
- Use the Clipboard
- Discuss the common Clipboard Terms
- Discuss the common Clipboard Pages

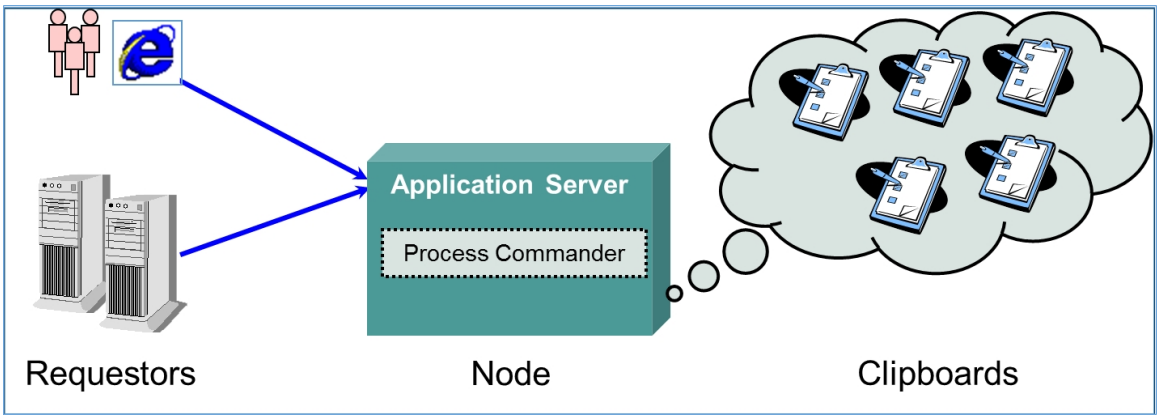
Things to Know

What is the Clipboard?

A requestor represents a person or process authenticated by PRPC. A guest user, an operator, or an external system are examples of requestors.

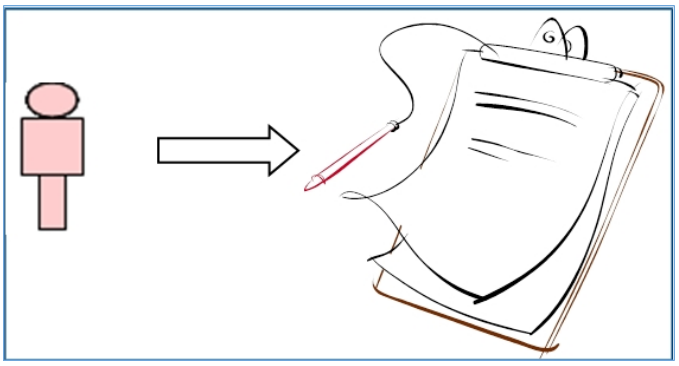
Every connected requestor (including every browser-based user, even if they are an unauthenticated guest user) has access to their own clipboard, a reserved, temporary memory area, allocated on the server upon authentication. Requestors are independent of each other and each has a unique identifier.

The clipboard has a hierarchical structure, consisting of nodes, known as pages that contain information about the server, requestors, and in-memory processes. A page provides a buffer or temporary copy of item instances (of that class) that are copied from, and may be stored in, the PegaRULES database or another database.



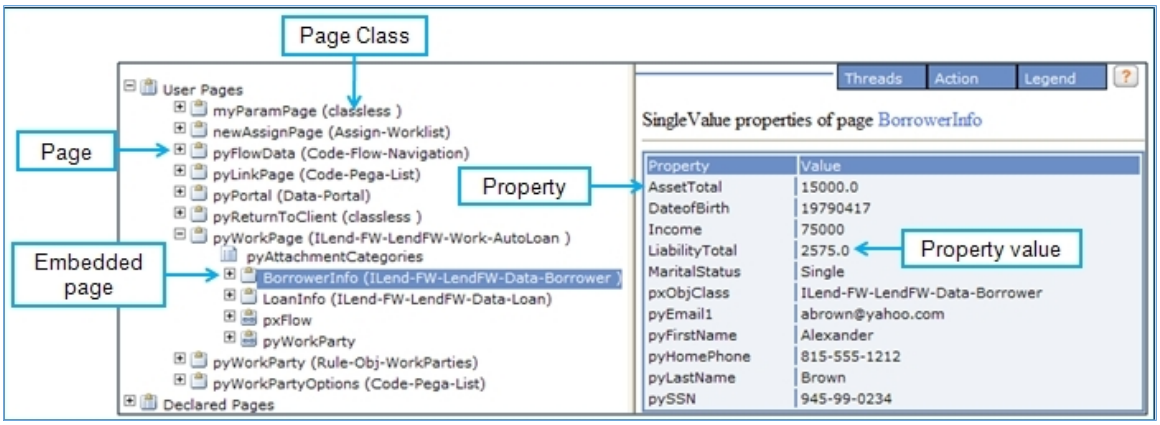
Using the Clipboard

The Clipboard tool allows developers to examine a snapshot of the structure and contents of their own clipboard and to troubleshoot issues by viewing and directly updating parts of their clipboard.



Using their workstation's browser, developers can access the Clipboard by clicking the Clipboard Tool icon on the Quick Launch toolbar. The clipboard consists of two panels. The left panel displays the pages that are currently in memory, organized as a tree structure. The right panel lists the properties and values set on the selected page, sorted by property name. Property messages appear in red text.

The left panel of the Clipboard tool may display page messages in red that convey error conditions, progress, or exceptions to a user. If the page messages indicates that the page (or a property on it) is invalid, check the value of a property. It may not meet the requirements of a permanent instance of the page's class, because of missing or incorrect data.



Common Clipboard Terms

A **node** represents a particular server within a cluster.

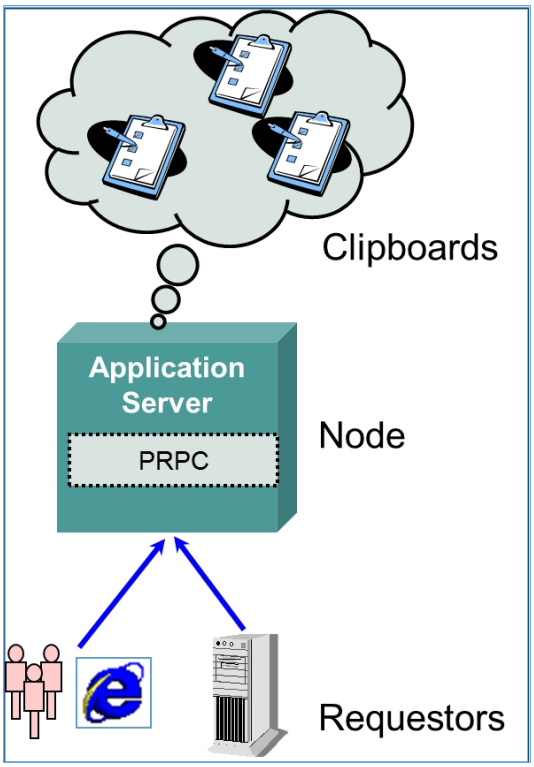
A **requestor** represents each entity that can query the node; each requestor is treated equally by the node. Each requestor is identified with a unique 32-character code, associated with threads representing their individual session.

Requestor type data instances define the types of requestors that can access PRPC, such as a browser session, an external application, or an internal background process. Each requestor type data instance determines the RuleSets, versions, access roles, and startup activities used by requestor sessions of that type.

Your requestor session has three main threads:

- **Standard** — a dedicated process, and single Thread ordinarily used to perform end-user work item processing for each interactive user accessing PRPC through a browser.
- **Developer** — Designer Studio session information.
- **Openportal** — Additional Thread started when you use certain tools, or open additional portal windows from the Designer Studio.

Examples of requestors include users or a SOAP service.



Common Clipboard Pages

Common clipboard pages include node-level pages and requestor-level pages. Node-level pages, such as **pxProcess**, typically contain node-specific information, like the machine name, host ID, and system port.

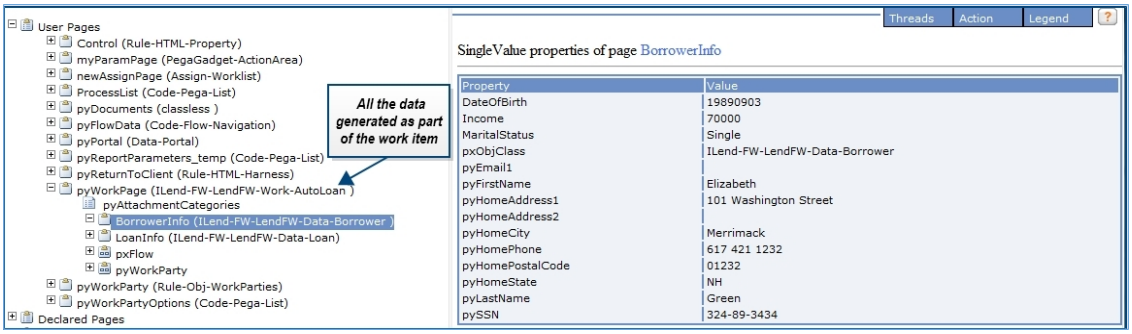
Requestor-level pages contain information about a requestor's session(s) and associated thread(s), the application they are accessing, its context, the operator and their organization.

- **pxRequestor** — contains information about the requestor's session, such as client name, IP address, access roles, RuleSet list, and HTTP protocol parameters. The system creates this page during log in.
- **OperatorID, Org, OrgDivision** — contains information about the operator, their organization, and their division.
- **pxThread** — contains information about the "context" of an individual application. Typically, each thread corresponds to a different browser window (or tab). For the Case Manager portal, each thread represents a tab in that portal: Dashboard, My Cases, Events, and Reports.
- **pyWorkPage** — the name of the page that contains information created during the processing of a work item when a single work item is opened onto the clipboard.

pyWorkPage

The pyWorkPage page is a common clipboard page that contains information created during the processing of a work item. When a new work item is created, its data is placed on the pyWorkPage page for that requestor. As the work item proceeds through its process, the contents of pyWorkPage, including any embedded pages it contains, are saved to the database.

As you work, clipboard pages related to your work item are created and updated by the system. Click **Refresh** from the **Action** menu to see these changes in the clipboard.

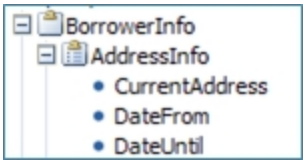


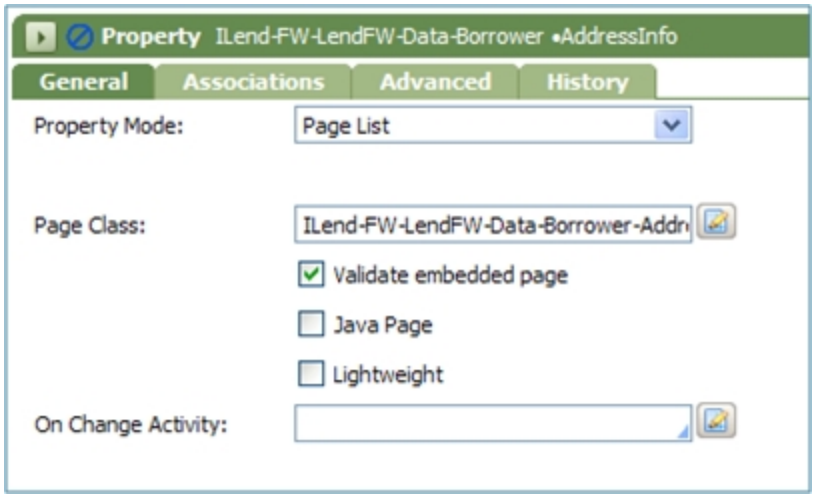
Other Clipboard Pages

Clipboard pages may be top level or embedded. Embedded pages are defined by a property of mode Page, Page List, or Page Group. These pages may, in turn, may contain other pages. The clipboard contains three broad categories of top-level pages: user pages, declared pages, and system-managed pages. The system creates many top-level pages in routine, normal operation. Your activities and data transforms can create other top-level pages. Most top-level pages are named, and have an associated class. While top level clipboard pages cannot be embedded, they may contain embedded pages.

Page properties allow creation of a data mode. Developers create page properties which act as data containers, and which can be associated with other page properties, and ultimately, with the work item. Page properties are defined by a name and an associated class.

Page properties can consist of an unordered list of unique property names and, optionally, their values. The properties contained in or inherited by the class define the properties which can exist on the page.





The name of a page may derive from any of four sources:

- Literal values entered into a rule, such as a parameter to the Page-New method. This creates a top level page.
- System created pages with reserved names and purposes, such as the pxRequestor page. This is a top level page.
- Names used by convention in a collection of rules designed to operate together.
- Property names (for embedded pages). This creates a page property.

Module 7: Reporting on Application Data

Lessons Covered:

- Reporting in PRPC
- Report Browser
- Report Definition Rules
- Accessing Data for Reports
- Viewing and Modifying Reports

Reporting in PRPC

Objectives

At the end of this lesson, you will be able to:

- Describe what reports do, and for whom they are useful
- Describe the report types available in PRPC
- Locate the standard report you want

Things to Know

Reporting Use Cases

The real purpose of reports is insight. You need ways of understanding how complex processes are functioning — where the bottlenecks are, where there are opportunities to improve response time, what emerging trends need attention. A report that asks the correct questions, and therefore provides you with relevant information rather than an unsorted heap of data, can show you what is going on now, what has been going on over a period of time, or how what is going on matches or differs from what was planned.

Anyone involved in your business processes, from the developers through the managers to the customer service representatives, will find occasions when the information a PRPC report can provide will be very valuable.

There are two common ways to use reports in PRPC:

1. Retrieve information and **embed** the resulting report in grids or other displays within forms that end-users use to create, update, and complete cases and work items. Embedded reports tend to run quickly because they retrieve a limited amount of relevant information, and run each time the form is opened.
2. Managers and analysts run **ad hoc** reports as needed to provide insight into organizational measures such as overall workload and status, organizational performance, key performance indicators, or trends over time in workload. Ad hoc reports often must summarize large amounts of information covering days, weeks, or months. Running a complex and extensive ad hoc report can have a performance impact on the database and the rest of the application.

Report Types Available

PRPC provides:

- **List-type** reports, which display requested information in a spreadsheet format, with each row representing a specific case, work item, product or other data type.
- **Summary-type** reports, which group data by start date, total purchase price, zip code, or some other factor common to all the rows in the report. Summary-type reports can include **charts** of various kinds to clarify the significance of the report data. For instance, **trend reports** can show data changes over some time period, and an accompanying line chart may help the viewer understand those changes.

Standard Reports

PRPC provides over 50 standard management reports to help business managers monitor and analyze processes or activities automated as PRPC applications. Developers access these reports in the Designer Studio; business managers access them through the **Report Browser** (see the "Report Browser" lesson).

The standard reports are grouped into categories:

- Monitor Processes
- Monitor Assignments
- Analyze Performance
- Analyze Quality

You can get insight into the types of ad hoc reports you might want by looking at the standard management reports shipped with PRPC. Developers should always consider using copies of existing reports as starting points for creating similar ad hoc reports to suit specific business needs.

For more information, you can access a listing and description of the standard reports included in PRPC through the Pega Developer Network in article 25863, "Listings and Descriptions of out-of-the-box reports".

Report Browser

Objectives

At the end of this lesson, you will be able to:

- Access the Report Browser
- Browse existing reports and search for a report
- Create your own categories for organizing reports
- Create shortcuts to existing reports
- Create a new report and save it to a personal category
- Schedule and subscribe to reports

Things to Know

PRPC users access and run reports from the Report Browser, which makes it easier for users to browse, search, organize, and run existing reports. Users can easily create their own reports, and share them with other users.

You do not need to be a developer to use the Report Browser, although you do need an unlocked production RuleSet version in which to save the reports and report categories you create.

Accessing the Report Browser

There are two standard ways to access the Report Browser. You can find it through the Designer Studio, and it is also part of the Case Manager portal and other composite portals.

Browsing and Searching for Reports

The Report Browser displays **shortcuts** to reports you can view and run. Report shortcuts are organized into categories, and you can generally find the report you want by looking in the category in which it is likely to appear. You can also search for a report, using words that may appear in its title.

Creating Categories

You can't create reports until you have created at least one personal category to put them in. You can have as many categories as you want to organize the reports you create and the report shortcuts you save.

There are two category types:

- **Personal** — personal categories, and the reports in them, are visible only to the operator who created them.
- **Shared** — If you can create Personal categories and create your own reports, you can create a Shared category and copy a shortcut to your report to that category. Others in your access group can see and use shortcuts in Shared categories.

Creating a New Report

When you create a new report in the Report Browser, it appears in the Report Viewer with a default set of columns. You can easily change the columns to display the data you need to complete the report.

See the lesson on "Viewing and Modifying Reports" for information on using the controls in the Report Viewer to structure the report to provide the information you want.

Scheduling Reports

You can schedule reports to run on a one-time or recurring basis, and either enter a list of users to receive the results through email or let other users subscribe themselves to scheduled reports of interest to them.

To schedule a report, right-click on a report and, in the window that appears, click **Schedule**.

Subscribing to Reports

When a report is scheduled and available to subscribe to, a **Scheduled** icon appears in its listing in the Report Browser. You can subscribe to the report by right-clicking its listing and clicking the **Subscribe** button in the window that appears.

The report is delivered to you via email each time it runs, or until you unsubscribe by right-clicking its listing and clicking the **Unsubscribe** button in the window that appears.

For more information, go to the Pega Developer Network and access article 26386 ("Reporting in V6.2 - Nine Tutorial Videos") which includes links to these videos:

- Using the Report Browser
- Scheduling Reports

Report Definition Rules

Objectives

At the end of this lesson, you will be able to:

- Explain the difference between working with Report Definition rules in the Report Browser and directly in the rule form
- Control the report's layout
- Specify and filter the data the report uses
- Use the Function Builder to provide the result of a function as the data for a column
- Use sub-reports to provide data to a main report
- Control user interactions with the report

Things to Know

In this lesson we'll cover working with Report Definition rule forms in the Designer Studio to create reports. To learn about creating Report Definition rules through the Report Browser, see the *Report Browser* lesson.

Working with Report Definition rules

We can work directly with Report Definition rules in the Designer Studio, or indirectly by accessing and editing the rule's report in the Report Browser and Report Viewer.

The benefits of working in the Report Browser and Viewer include:

- A manager, without developer access, can quickly and easily create reports.
- Users can share with colleagues, schedule, and subscribe to reports through the Browser.

When we work directly in the Report Definition rule, however, we can define and control the report in ways not available in the Report Browser and Viewer, including:

- Allowing the report to appear in the Report Browser
- Controlling how Report Viewers can interact with the report
- Using sub-reports to filter or provide data to the main report

Control the Report's Layout

You set the appearance of the report in the **Design** tab of the Report Definition rule form. The tab includes four sections:

Columns To Include

Your report can have as many columns as you like. However, a focused report displaying the relevant data that best answers a question tends to be more useful than a too-comprehensive report where the reader can get confused while working through all the data.

You can add columns and adjust their order and the formatting of their content. You can also use the values in certain columns to group the values in other columns.

The value displayed in each column can include the:

- Value of the property identified with that column in the **Column Name** field
- Result of a summary function, such as COUNT() or AVG(), performed on the column's property

- Result of an SQL expression assembled using the Expression Builder (see below)

Rows To Include

You can filter the report's data so it only displays data that satisfies your filter conditions ("status equals open" and "customer ID equals 'WidgetCo'").

You can have as many filters as you need, and each filter must appear in the statement you create in the Filter Conditions field, like "A AND B".

Top/Bottom Results

For some reports you may want to limit the number of items that match the report criteria best (or worst). You can set the report to display the top- or bottom-ranked data, based on a data field of your choice.

General Report Settings

Beyond setting the report title, use this section to set certain display options:

- **Header Display** — If the report header displays, users can right-click on column headers and perform certain functions using the context menu that appears. See the lesson "Running Reports" for more information.
- **Custom sections —** You can create custom sections to display rows in the report or the filter conditions the report uses. This is useful when you want to make the best use of the monitor display space and cut down on how much the viewer has to scroll left and right to see the data about a single row in the report.
- **Display in the Report Browser** — Select this checkbox to make the report available in the Report Browser, where you can edit it and share it with colleagues. See the lesson "Report Browser" for the advantages of selecting this option.

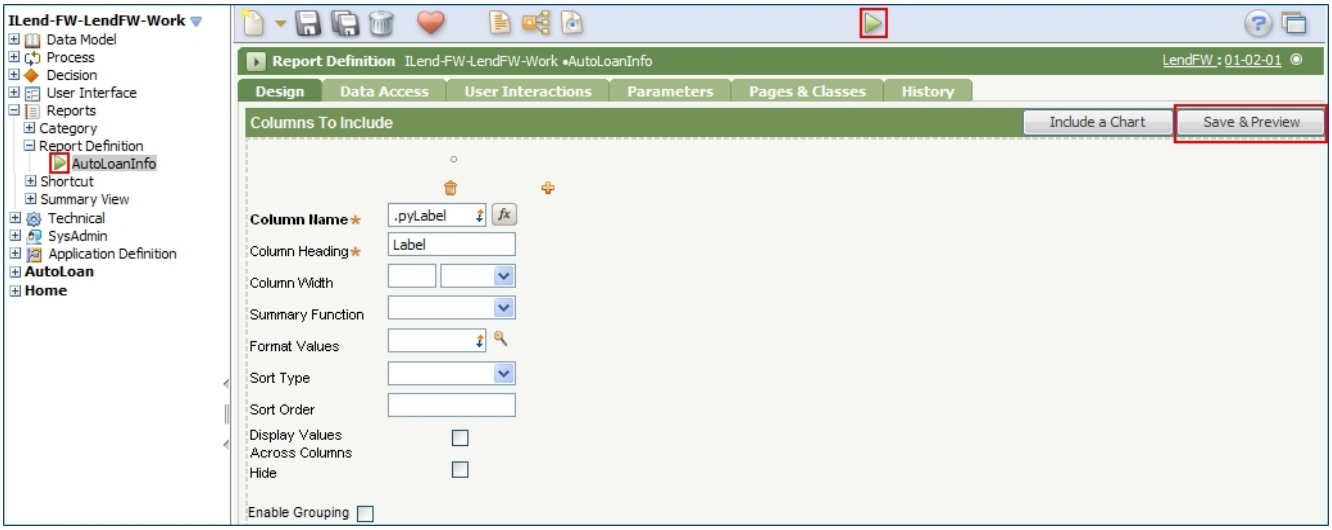
Specify and Filter the Data the Report Uses

Filter conditions limit the amount of data returned to the report ("only orders made by these customers", "only bug reports with a severity higher than 60", and so on).

Testing Reports

PRPC provides three ways for us to test our reports as we're creating them. To run our reports from Designer Studio, we can:

1. Click the Run button on the toolbar, as for any other rule type.
2. Click the Run icon that appears next to the report's name in the Application Explorer. This allows us to run a report without first opening the rule form.
3. Click **Save and View** on the report definition rule form. This saves the report first, and then runs it in the Report Viewer. This option is most helpful when developing a report, to incrementally test your report as you create it.



Use the Function Builder

On the **Design** tab, to the right of the **Column Name** field for each column, is an icon that, when clicked, displays the Function Builder in a pop-up window. The Function Builder helps us select a SQL function template to use as the basis for the SQL function we wish to use. When we have selected the template, the form displays further fields where we select the data the function is to use. For instance, if we want the column to display quarters (three-month sections of the year), we could use the "first day of the quarter of [Date Value]" function, and provide the data property that holds the date (last updated on, created on, or whatever) that is of interest for the report. The function returns the quarter of the year the date supplied falls in.

We can specify another SQL function as a property for a value in a SQL function. The function builder provides fields and SmartPrompt assistance wherever this is possible. Note, however, that application performance may degrade if we specify SQL functions within SQL functions beyond five depth layers.

Use Sub-reports to Provide Data to a Main Report

On the Data Access tab we can identify other Report Definition reports to serve as sub-reports to the report on which we are working. Sub-reports can provide data to the main report, or contribute to the filter conditions we establish.

Sub-reports are completely functional reports and can run in their own right. We cannot use as a sub-report a report that already has a sub-report or sub-reports of its own.

Control User Interactions with the Report

As noted above, we set several options that affect user interactions on the Design tab. There are even more on the User Interactions tab. Most are enabled by default, but here is where we establish other options such as:

- Which icons appear in the command ribbon of the Report Viewer for this report. See the lesson "Running Reports" for more about the Report Viewer.
- Whether the Report Viewer can alter the filter conditions "on the fly".
- Whether the report is divided into pages that fit better on a monitor display, or appears all at once in a display the viewer must scroll through.
- Whether SmartInfo windows appear to provide more information about the data.

For more information, see the Report Definition Rules topic in the Developer Help, available in the Designer Studio.

Accessing Data for Reports

Objectives

At the end of this lesson, you will be able to:

- Explain how class data is stored in the PRPC database
- Explain the implications of PRPC data storage for reporting
- Access all the data needed for a report

Things to Know

PRPC initially stores the data entered or displayed in application user interface forms in memory, on what's known as the clipboard (which is not visible to the end user). When an end user clicks OK or Submit to save changes, the system writes the data to the database.

The data for a class instance representing a case or work item is stored in a single row or record of the database table for that class. However, the data model for a case or work item may be quite large and complex, including multiple levels of hierarchically embedded sets of fields (PRPC refers to these sets of fields as **pages**, **page lists**, and **page groups**).

Most of the database operations that PRPC performs, such as opening a case or work item from a worklist, saving changes to a case or work item, or reading a business rule, are optimized through PRPC's architecture which uses a single **binary large object** (BLOB) field within the record to store all of the data for the case or work item. This storage method, while highly efficient, poses two issues for reporting:

1. When reports assemble data for display, or try to return a specific work item in response to a search request, PRPC provides certain "exposed" properties for each work item, not all the data stored in the BLOB.
2. Reports often rely on database queries using SQL (structured query language) statements, and PRPC reports generate SQL statements using a convenient interface. however, the information in the BLOB field is not directly accessible within SQL statements.

For reporting, PRPC can:

- Expose (add) top-level BLOB properties to the class table as database columns.
- Store BLOB properties in embedded pages, page lists, and page groups in columns in separate tables called **declarative indices**. You can join rows in the declarative indices to their corresponding class tables to make them available for use in reports.

Exposing columns and building declarative indices is a straightforward process in PRPC, but it involves a change to the database schema. The benefit of keeping data in the BLOB is that it maximizes flexibility during the development of the application. Exposing columns makes their data available for reports, but reduces design flexibility.

Consult with your database administrator or system architect, who may make the changes for you or direct you to do them.

Note: You can expose a property even after the property has values. However, it may take some time for the system to copy existing values from the BLOB into the new column.

The Property Optimization tool enables you to make changes to the database schema. It is available only on development systems — systems where the production level is 1 or 2.

Using Exposed Properties in Reports

Top-level properties that have been exposed, and embedded properties for which declarative indices have been created, are available for use in reports. They appear among the autocomplete choices for a report, wherever you can:

- select columns for display
- specify parameters in SQL functions
- specify properties in filter conditions

For more information, see the lessons "Report Definition Rules" and "Viewing and Modifying Reports" for more information on using properties in reports.

Viewing and Modifying Reports

Objectives

At the end of this lesson, you will be able to:

- Use the editing controls in the Report Viewer
- Apply filter conditions to a report
- Add a chart to a report
- Use the column heading (right-click or context) menu

Things to Know

Using the Report Viewer Taskbar

Click one of the icons on the Report Viewer taskbar to interact with the report. The ways you can interact with the report are listed below. Some or all of these features may be disabled, based on settings within the report or on your access privileges.

- Save
- Save a copy
- Print
- Edit the columns to include or what data they represent
- Add columns
- Add or update a chart
- Export as a spreadsheet or as a PDF document
- Filter the report's data

Using the Report Editor

The Report Editor, extending the Report Viewer, makes it easy for managers or developers to add and delete report columns, add or update computations, and make many other changes, without using the Report Definition form. It displays the report you selected, or the new report you created, and provides an array of tools and resources to help you edit it.

Accessed by clicking the Edit Report  icon in the Report Viewer, the Report Editor has several features:

- The **Data Explorer** provides a quick way to find a property or calculation with which to populate a column in the report. Within the Data Explorer, there are three tabs:
  - The **Best Bets** tab displays the properties that you are most likely to use in your report, organized in a tree structure.
  - The **All Matches** tab displays a broader list of the properties you are most likely to use in your report, organized in a tree structure.
  - The **Calculations** tab allows you to select a SQL function and identify one or more properties for it to work with.
- The **toolbar** that appears at the top of the right panel of the report editor contains a different set of controls depending on whether you are working with a list-type or summary-type report. For a complete listing of these controls, see the Developer Help.
- A series of **Hints** appear just below the toolbar to help you understand the various editing options and what you should do next.

Applying Filter Conditions

Filter conditions limit the amount of data returned to the report ("only orders made by these customers", "only bug reports with a severity higher than 60", and so on).

You can set filter conditions in three ways:

- If the developer has opted to display the report header, a statement of the logic of the active conditions appears there in the form "Filters: Urgency >= 50". The statement will be as long as necessary to include all the filter conditions. Filter conditions that the report user can edit appear as hyperlinks. Users can click a link to see the Report Filter form. The existing filter condition displays, and users can update the elements of the filter (see below).
- If the Advanced Filtering control appears on the Control Ribbon, you can click it to display the Advanced Filtering form. You can then, depending on the settings for the report, modify or delete existing filter conditions, add others, and modify the logic statement that connects the filter conditions.
- If the column headers appear, you can right-click on a column header and select "Advanced Filtering" from the context menu. See below.

Adding a Chart

For a summary-type report you can add a chart to display the report data graphically. A wide range of chart types is available, including:

- Pie
- Bar and Column
- Area
- Line

For each chart type, the **Add or Edit Chart** dialog lets you select the data from the report that the chart is to use. Experiment with the options and settings in this dialog to create a chart that best conveys the report's information.

Note that some chart types are better than others for depicting certain types of data. For instance, use a line chart to graph a trend report that depicts changes in data over time.

Using the Column Heading (right-click) Menu

If the report's column headers appear, you can right-click on any column header and select any of the available commands.

- Hide the current column.
- Create a summary view of a list-type report, using the current column's values to group the report data. (Note: this option may not be available or may not generate useful results for all columns, such as columns holding unique IDs.)
- Adjust data filtering for the report. When you select "Advanced Filtering" a form appears that displays the current filter conditions. You can modify or add to them, or change the logic that relates the rows of filter conditions.

For more information, go to the Pega Developer Network, and see article 26386 ("Reporting in V6.2 - Nine Tutorial Videos") which includes links to these videos :

- Report Viewer Basics
- Changing the Rows Included in a Report
- Selecting Columns and Entering Calculations
- Changing Column Order and Column Format
- Working with Summary Reports
- Working with Charts

Module 8: Completing the Process Definition

Lessons Covered:

- Create Flows
- Screen Flows
- Work Status
- Routing Work
- Service Levels

Creating Flows

Objectives

At the end of this lesson, you should be able to:

- Create new flows
- Configure a flow to be a starting flow
- Call a sub flow

Things to Know

Flows are rules that represent business processes or parts of a business process. As we implement PRPC applications, we may create many flows to encompass an entire process. In this lesson, we discuss the types of flows allowed in PRPC, and how to use them.

Flow Types

PRPC provides three ways to use flows. Note that each usage is represented in PRPC as an *instance* of the **flow** rule type.

Starter flows

A starter flow is used to create work items. Starter flows appear in the Run menu, and are indicated in the Application Explorer with a Run  icon.

Sub flows

A sub flow is a subprocess. Sub flows are used to collect common flow steps, either for reuse (they represent tasks performed in multiple processes) or to reduce flow complexity (by reducing the number of shapes on the flow diagram). Sub flows can only be called from another flow, either a starter flow or another sub flow.

Screen flows

A screen flow is a special type of sub flow, intended to present a sequence of forms to a single operator. In a screen flow, operators have only one option for advancing the work item through the process, and ownership of that item cannot be transferred to another operator. As a result, fewer shapes are available when editing a screen flow. Screen flows have other special characteristics, which we'll cover in another lesson.

A screen flow **cannot** be used as a starter flow.

Calling a Sub Flow

We can use several flow editor shapes to call a sub flow from another flow, the most-commonly used is the **subprocess** shape.

The subprocess shape includes several important attributes that affect how it processes work. They are:

Define Flow

We can choose to run the flow on the current work item, on another work item, or on the contents of an embedded page.

Filter Flow Rule By

We can specify that the sub flow is either a regular flow or a screen flow. If the calling flow is a screen flow, the called flow **must also** be a screen flow.

Spinoff Flow

By default, sub flows perform a set of tasks and then return control to the calling flow. In certain circumstances, however, we instead want to perform those tasks *asynchronously*. To do this, we can "spin off" the sub flow as a separate process.

Flow Attributes

PRPC provides options to control the execution of flow rules. These options may draw upon other rules, and are set on the **Process** tab of the flow rule.

Creates a new work object?

The most important option for a flow rule is **Creates a new work object?**, which differentiates a starter flow from a sub flow. When enabled on a flow, the flow can be used as a starter flow. Enabling this option also provides other options, such as the ability to:

- Specify a default harness when creating the work item. The default value is the **New** harness. (We'll cover harnesses in another lesson.)
- Skip this creation step altogether and proceeding directly to the first step in the flow.
- Specify a data transform rule to provide initial property values.

Other Options

Other notable options on the Process tab are the ability to:

- Access other pending assignments once the operator has completed their portion of the process.
- Document the process as a starting process when using the Application Documentation wizard.
- Set security to control which operators can run the process, and under what circumstances.

Screen Flows

Objectives

At the end of this lesson, you will be able to:

- Describe the benefits of using a screen flow
- Describe the difference between the types of flows
- Create a screen flow to sequence operator assignments

Things to Know

In this lesson, we are going learn more about a specific type of flow — the **screen flow**. Screen flows have the following characteristics:

- They sequence a set of assignments performed by one operator.
- They only allow one option for advancing the work item through the process.
- They cannot be used to create a work item.

In essence, a screen flow can be thought of as one large assignment, separated into multiple forms. Consider the following example:

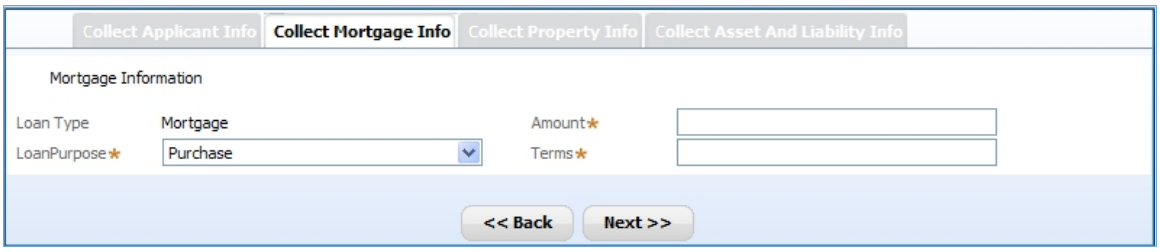
If a customer calls an insurance company to file a claim, the customer service representative (CSR) must collect basic information from the customer — such as name and policy number — before collecting details about the claim itself. These tasks *could* be represented in the process with one assignment. However, for ease-of-use, they are usually separated into multiple assignments, with a separate form for each assignment. Since all of these assignments are performed by the CSR, before the claim is filed and routed for review, they can be combined into a screen flow.

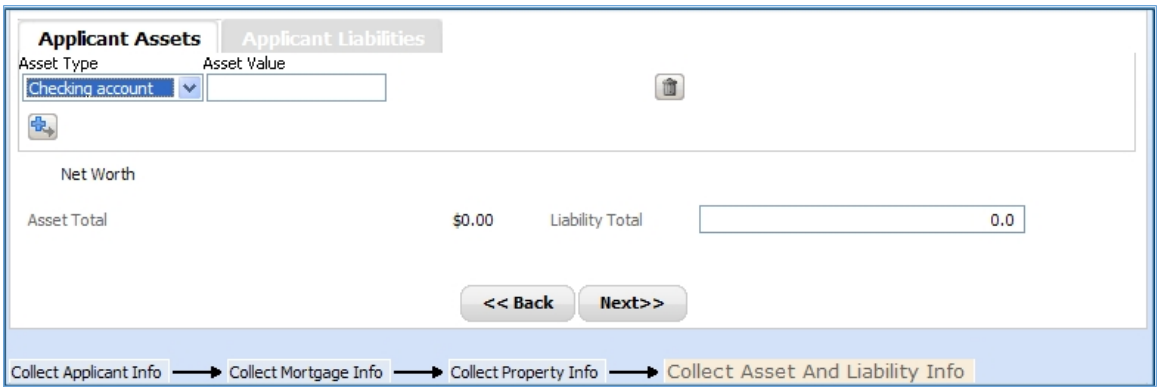
Benefits of a Screen Flow

In PRPC, screen flows automatically provide controls to facilitate navigation (backward and forward) between assignments. This means that:

- Operators are aware of and understand the sequence of assignments.
- Operators can tell where they are within that sequence.
- Operators can easily navigate backward to make changes, then return to the current assignment.

The following screen flow examples illustrate how this navigation may appear to an operator. Note the navigation controls — the tabs in the top example, and the breadcrumbs in the bottom example — that guide the operator through the sequence of assignments.





Differences Between Screen Flows and Regular Flows

Before we create a screen flow, there are some important differences between screen flows and regular flows that we should discuss.

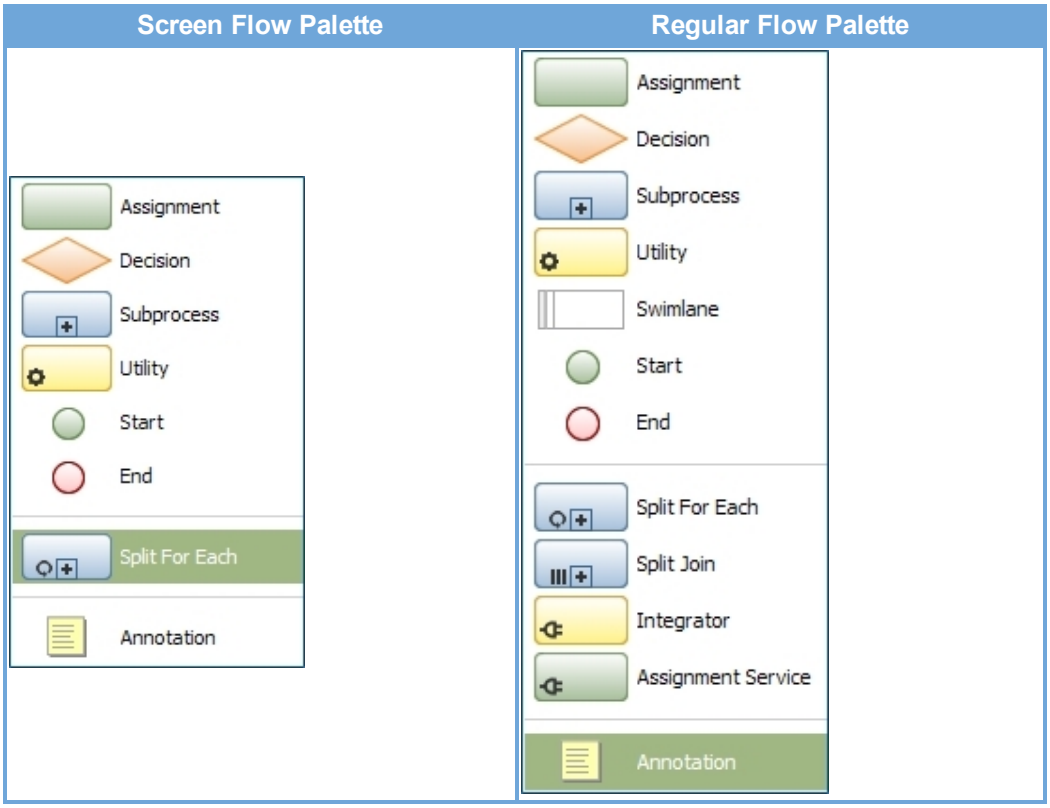
Flows Must Be Created as Screen Flows

With regular flows, we can use a starter flow as a sub flow. We can also enable or disable the option to create a work item, which distinguishes a flow as a starter flow. In short, regular flows are flexible, and can be configured as they're used.

In contrast, screen flows *must* be created as such. We cannot convert a flow into a screen flow after its creation.

Screen Flows Provide Fewer Shapes Than Regular Flows

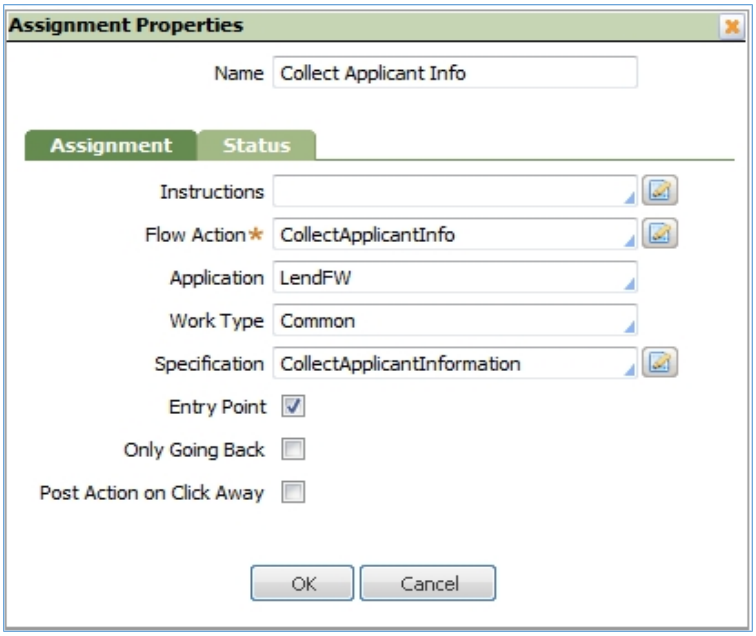
When we create a screen flow, PRPC requires us to specify a *screen flow template*. Our template choice affects the shapes available for use in the flow. Screen flows utilize a reduced set of shapes, omitting the advanced flow shapes that pass control to other actors (either operators or systems).



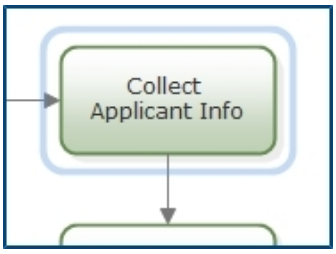
In Screen Flows, Flow Actions are Associated with Assignments, not Connectors

As we mentioned earlier, screen flows only allow one option for advancing a work item off of an assignment. This occurs because, in a screen flow, flow actions are specified on an assignment, rather than a connector. Since flow actions represent the possible options for advancement, and only one flow action can be specified on each assignment, we're reduced to one possible path for advancement.

In the following example, note how the assignment shape in a screen flow requires that we specify a flow action.



And on the flow diagram, note how the connector no longer references a flow action.



Work Status

Objectives

At the end of this lesson, you will be able to:

- Explain some common work item status values and their significance.
- Identify the property that holds the status value.

Things to Know

What is Work Status?

Work status denotes **the current status of a work item**, and changes as a work item progresses through a flow. When all work on a work item is complete, it is considered **resolved**, or closed. At this point the work item no longer appears on any worklist or in any workbasket and does not require any more processing, but all work item details remain stored in the database.

The pyStatusWork property holds work item status.

An **open** work item is one that is *not yet resolved*, or one that has been *reopened* after it was previously resolved. An open work item requires additional automated or interactive processing. You cannot work on (modify) a resolved work item; it must first be reopened.

As a work item progresses from initial entry through resolution, the status value may change only a few times, or dozens of times. It is important that the status value, often used in management reporting, reflect accurately the condition of the work item.

How to Set Work Status

Standard Work Status Values

Standard work status values begin with:

- New — Not yet reviewed or qualified
- Open — Responsibility for processing is within the processing organization
- Pending — Responsibility for processing is currently with an external organization
- Resolved — Processing is complete

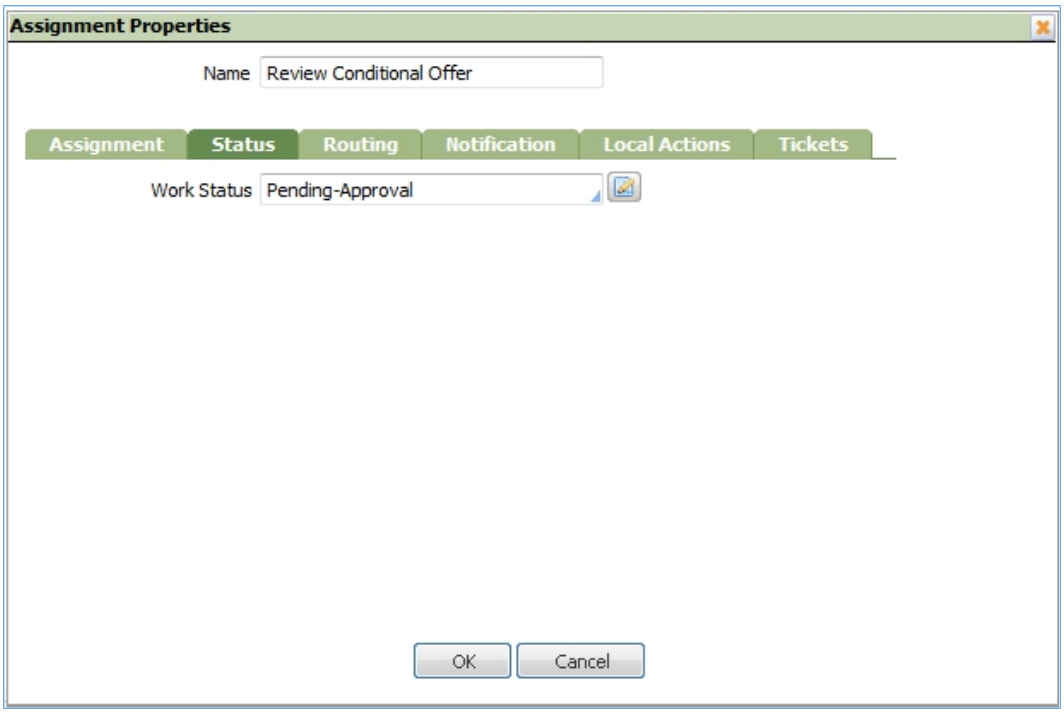
Typically used values are New, Open, Pending-Approval, Resolved-Completed, and Resolved-Rejected.

Your application can include additional status values. When creating custom status values, always use one of the four keywords above as a prefix; for example:

- Pending-Audit
- Resolved-Closed

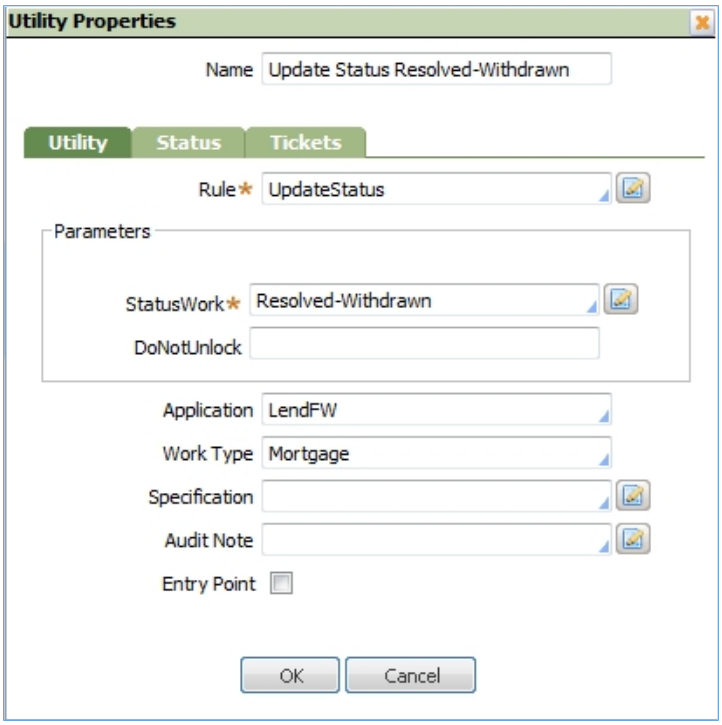
Updating Work Status on an Assignment

Your application can automatically update the status when the work item reaches an assignment shape within the flow. To use this method, specify the StatusWork value within the Assignment properties.



Setting Work Status with a Utility

Use a utility shape to call the standard UpdateStatus activity, which will then run when that point in the flow is reached. Specify a value for the StatusWork parameter, using the Properties panel.



Routing Work

Objectives

At the end of this lesson, you will be able to:

- Explain Routing to a Worklist or Workbasket
- Explain Operator Availability
- Describe Skill-based routing
- Describe Routing with a Decision Rule
- Explain How Routing Leverages the Organizational Structure

Things to Know

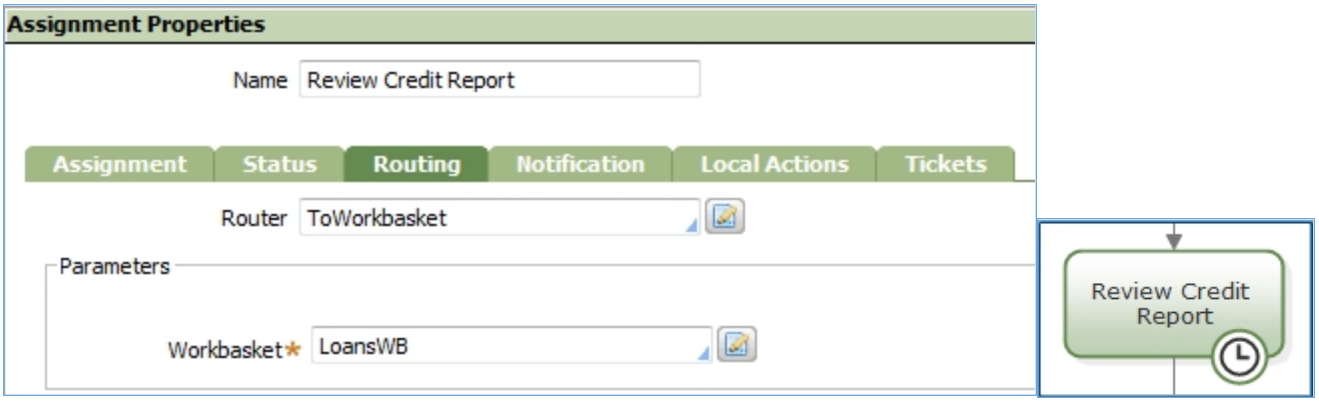
What is Routing?

Routing is the process of moving work through a system. There are two industry-wide models used to route work: push and pull. The push model is used when work is sent to an operator, while the pull model defines scenarios where operators select work from a workbasket. As a flow rule executes, PRPC can create assignments and route the work by placing it in an appropriate worklist or workbasket so that a human user, an agent, or external participant can perform their assignment to process the work.

Determining who is to process each assignment — whether automated or through a manual decision — is often an important factor in the throughput of the business process and the business success of an application.

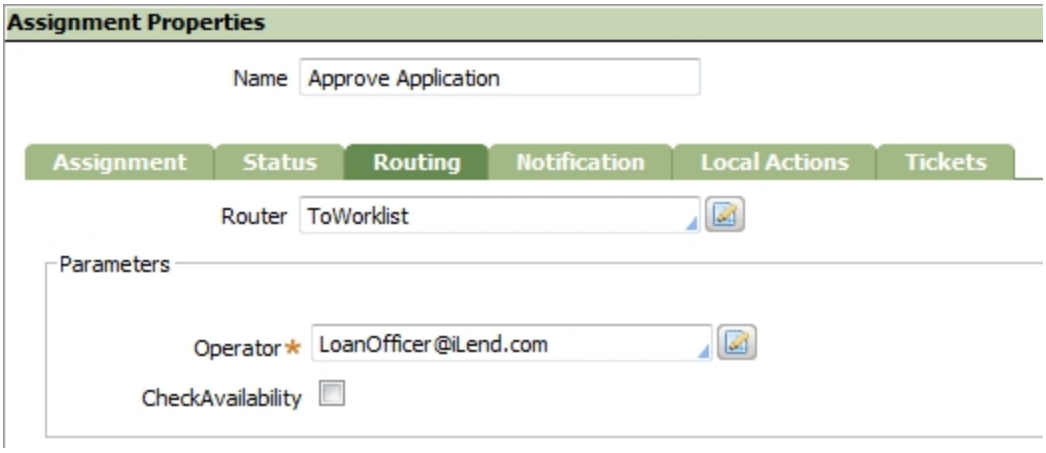
Routing to a Workbasket

An assignment can be routed to a workbasket rather than to a worklist. Users who are assigned to that workbasket can access the assignments that have been routed there, as well as the assignments on their worklist. When the assignment is processed it disappears from every worklist that is “fed” by the workbasket.

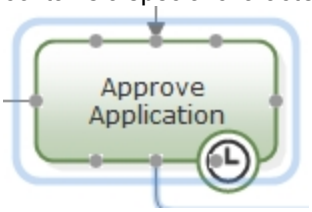


Routing to a Worklist

As a general rule, allow PRPC to dynamically look up and route assignments to users, instead of assigning work to a fixed, individual user. However, if certain work items need to be routed to a specific user, configure the assignment to route the work to that individual. The Operator field must include quotes around a value such as LoanOfficer@ilend.com because it

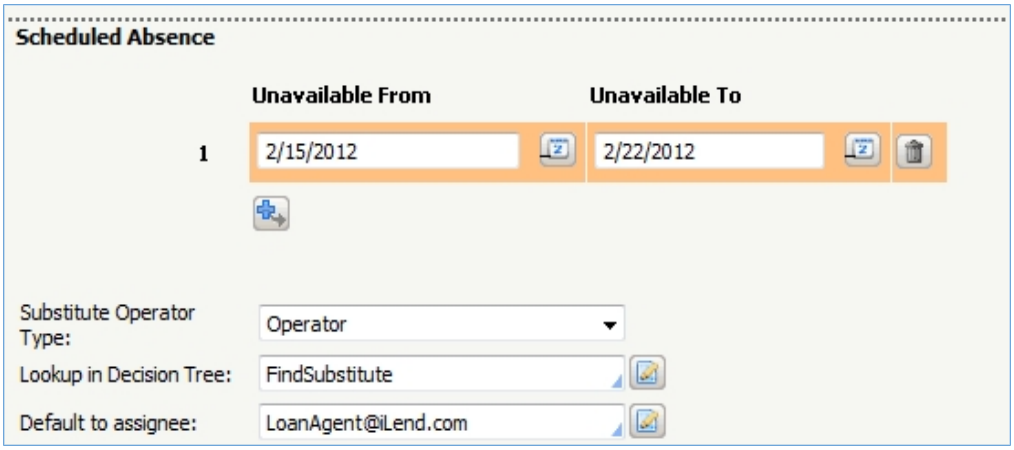


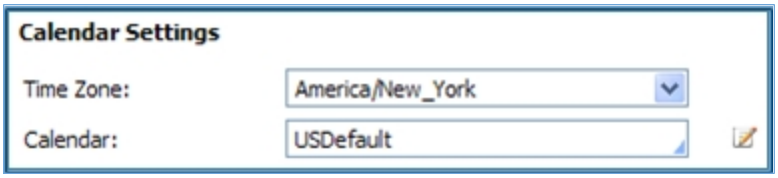
contains a special character.



Operator Availability

The **Work Settings** tab (shown below) on the Operator ID record lets us specify schedules, scheduled absences, and a substitute (backup) operator or workbasket. This enables us to assign work to a substitute operator if the assignment is to be completed during a period when the first operator selected is unavailable. The substitute can be a specific operator or workbasket, or it can be determined by a decision tree rule.





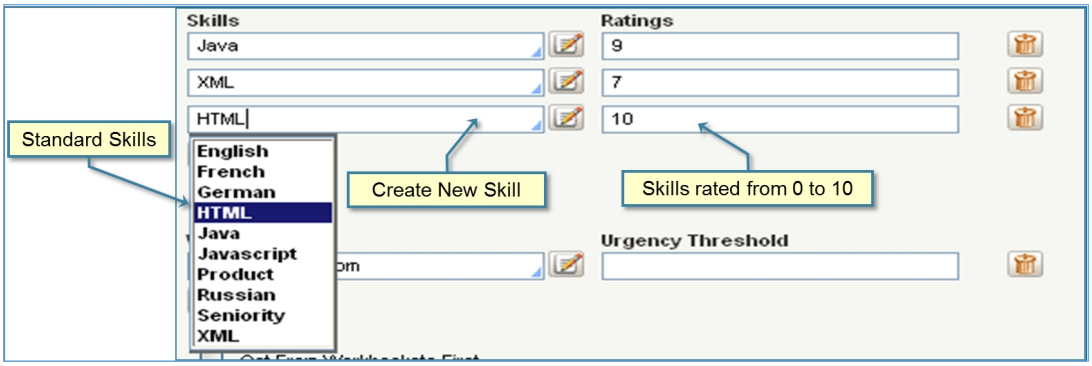
Use Information about an Operator's Skills to Route Assignments

PRPC's intelligent routing algorithm can examine backlogs, the availability or absence of operators, operator skills, the urgency or priority of an assignment, a customer's location or time zone, and other factors.

Skill-based routing, an important type of intelligent routing, examines user authority and urgency of the work item, and compares the characteristics and skill proficiency of operators in a work group with the characteristics required by the new assignment, to make a good or best-match assignment. For example, if an assignment requires an operator to speak Spanish proficiently, that assignment will be routed to a person who has identified Spanish speaking as a skill proficiency.

Routing activities and security tests use skill information for intelligent routing of assignments to those operators who meet minimum qualifications, or to the most qualified worker. This can significantly affect the productivity of an entire work group. (To reduce processing demand, avoid this approach in situations where work groups may contain hundreds of operators). Much of the sorting and selection occurs through efficient SQL operations on the PegaRULES database.

The **Work Settings** tab on the operator ID record lets us specify the operator's skills that can be used to match against the skill requirements of assignments. The screen shot below shows how we can assign standard skills to operators or create new skills.



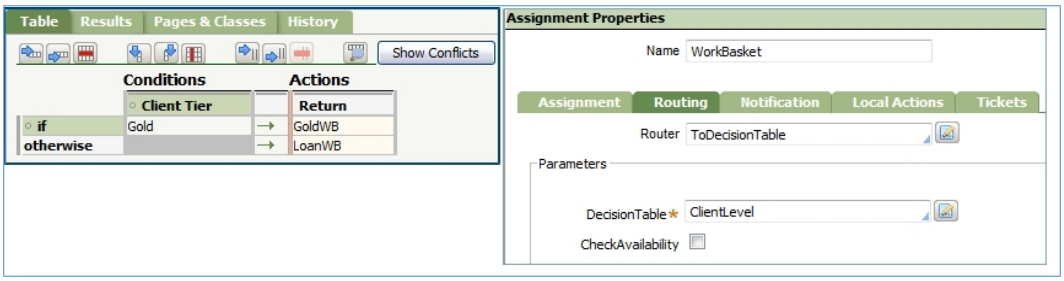
Router	Description
ToSkilledGroup	Routes the assignment to a randomly selected operator in the work group, based on availability and skill. If no operator is available, the assignment is routed to the manager.
ToSkilledWorkbasket	Routes the assignment to the specified workbasket and assigns skills for this assignment. When retrieving work from a workbasket, the operator must have the requisite skills to work on the assignment.
ToLeveledGroup	Routes the assignment to a work group whose operators have the required skills and are available, then chooses the operator who has the least urgent work.

Routing with a Decision Rule

A decision rule can be used to route an assignment. This is done by specifying one of the standard router activities, such as ToDecisionTable and ToDecisionTree, on the **Routing** tab of the Assignment shape in Process Modeler, as shown below. The routing activity name, specified as a parameter, is the name of the decision table or tree. The routing activity also checks operator availability.

Router	Description
ToDecisionTable	Routes the assignment based on the result of a decision table.
ToDecisionTree	Routes the assignment based on the result of a decision tree.

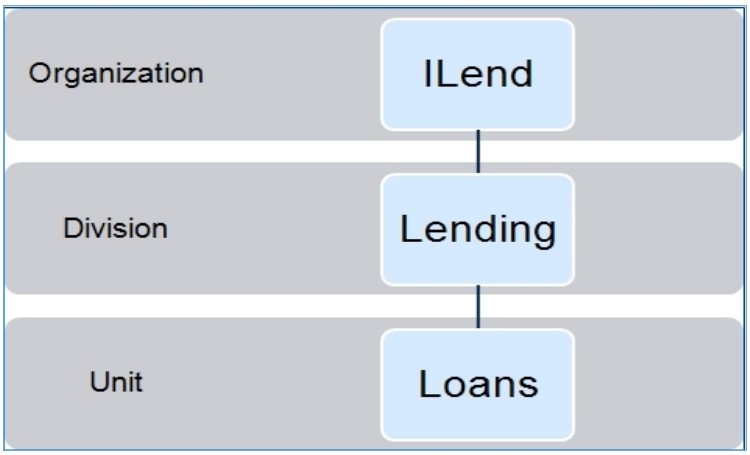
For example, using a decision table rule, assignments involving customers with specific risk levels could be routed to individual operators who are familiar with the requirements for that type of customer.



How Routing Leverages the Organization Structure

PRPC allows developers to model their company's organizational structure. Once the organizational structure has been defined, assignments can be routed to users at different organizational levels (organization, OrgDivision, or OrgUnit).

For example, given the ILend company's organizational structure (below), assignments can be routed to users, managers, and work groups within the ILend organization, the Lending division, or the Loans unit. (When developing applications, you may notice an additional unit named TestUnit. This unit is created by the AA for development purposes.)



We can use the Organizational Chart to view and edit the organization hierarchy. The Organizational Chart is available on the Organization landing page.

Routing Rules That Leverage the Organizational Model

Below is a list of a few of the many standard routing activities that are available. System Architects can create additional routing rules.

Rule	Function
ToOrgUnitManager	Routes to the manager of the organizational unit

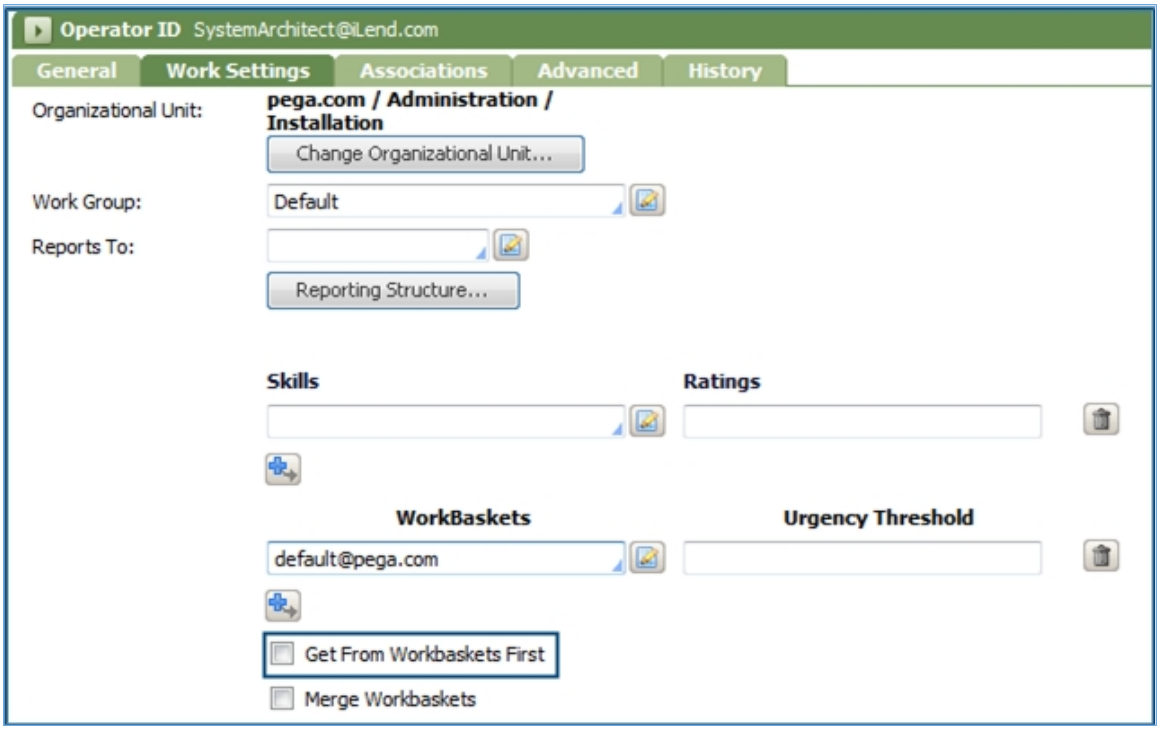
Rule	Function
ToCostCenterManager	Routes to the cost center manager for the division
ToWorkGroup	Routes to the operator's work group
ToWorkGroupManager	Routes to the work group manager

Get Most Urgent

Each time a user completes an assignment, an application can automatically provide another assignment for that user to work on next. The order that users process work may affect user productivity, processing timeliness, and customer satisfaction. If an application does not provide automated support for this function, users can arbitrarily or intentionally choose their next assignment from those visible to them.

In most situations, users are directed to click the **Get Most Urgent** button when it appears in their user portal. This button starts an activity that can search through the contents of worklists, workbaskets, and apply many factors to determine which assignment is best for the user to work on next. Get Most Urgent is basically a combination of the push and pull models.

We can also add additional workbaskets to the user's operator ID form. This allows the user to receive assignments from workbaskets other than the one associated with their workgroup. On the Operator ID form, we can select the **Get From Workbaskets First** checkbox to force the system to look in these additional workbaskets for the next assignment before it looks in the user's worklist.



Service Levels

Objectives

At the end of this lesson, you will be able to:

- Create a service level rule and associate it with an assignment
- Describe how response time is calculated
- Describe how urgency is calculated

Things to Know

Service Level Processing

Service level processing is a mechanism to ensure that users perform work in a timely fashion. A service level indicates the expected or targeted turnaround time for an assignment or work item, providing metrics or standards for the business process. Service levels define three time-based events, specified in days, hours, minutes, and seconds:

- **Goal** — The time before which the work should **ideally** be completed. This interval is shorter than the Deadline interval.
- **Deadline** — The time before which the work **must** be completed. This interval is longer than the Goal interval.
- **Passed Deadline** — The assignment has exceeded the specified deadline time, and is now considered **late**. (Passed Deadline is only available for assignments).

Service-level monitoring of work is performed by a system agent. As work remains open beyond each interval, PRPC flags the work to draw attention to it. This allows users to prioritize the completion of overdue work. Assignments are indicated in a user's worklist with color-coded entries, and work items are indicated on the My Cases tab of the Case Manager portal.

In the **Case Worker** portal, the worklist display is initially sorted by urgency, with highest urgency appearing first.

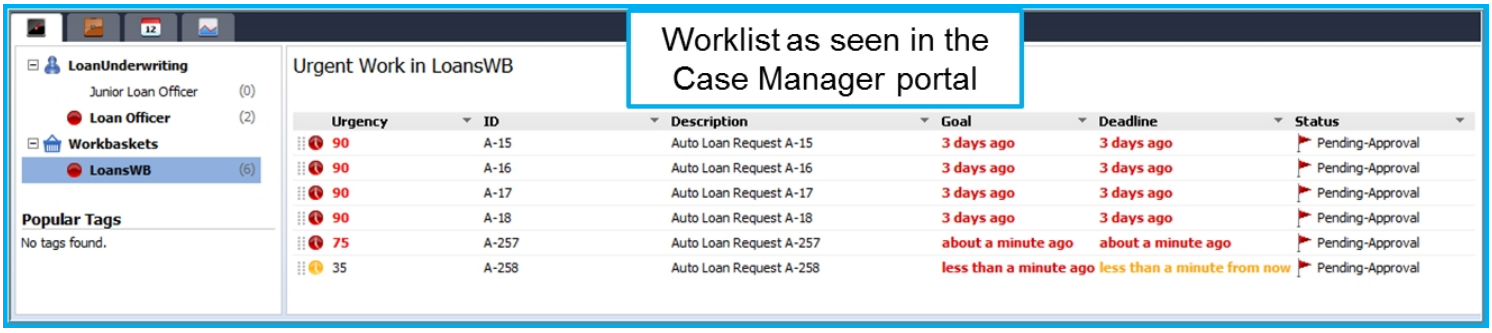
- A red alarm clock indicates that the assignment is within one hour of the deadline.
- A yellow alarm clock indicates that the assignment is within one day of the goal.

Urgent Work

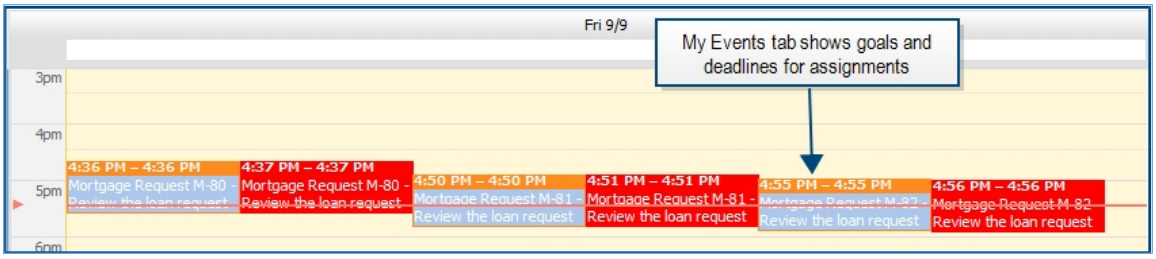
Worklist as seen in the Case Worker portal

Urgency	ID	Description	Goal	Deadline	Status
90	A-258	Auto Loan Request A-258	40 minutes ago	39 minutes ago	Pending-Approval
35	A-259	Auto Loan Request A-259	less than a minute ago	less than a minute from now	Pending-Approval

In the **Case Manager** portal,



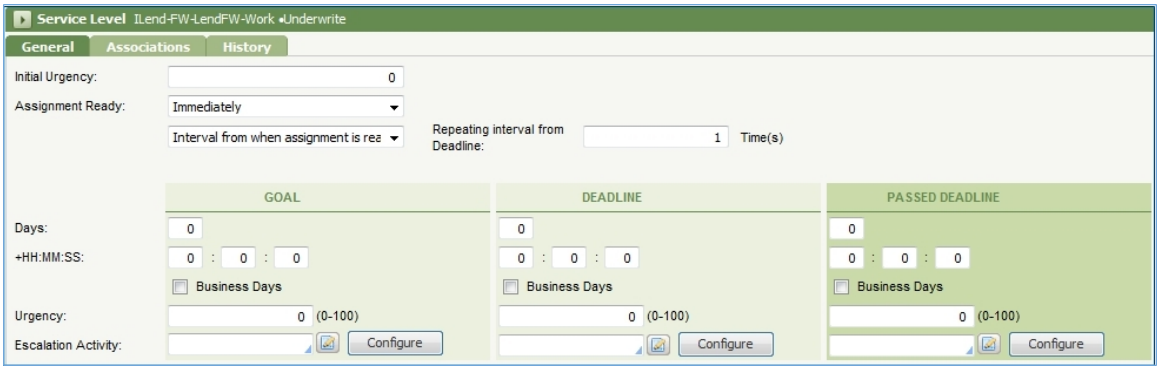
- An assignment that has passed a goal appears as orange on the My Events calendar tab.
- An assignment that has passed a deadline appears as red.



To apply a service level to an assignment, we use a service level rule.

Associating a Service Level with an Assignment

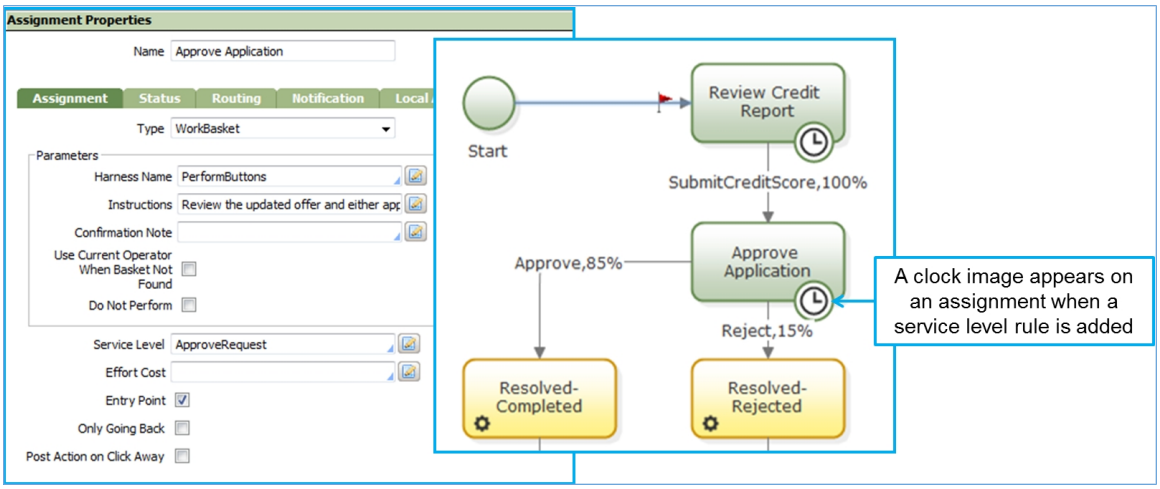
Users who receive an assignment are expected to complete it before it reaches the goal time interval defined by the service level. If the assignment has not been resolved when the goal or deadline time interval is reached, then the assignment may become more urgent and any escalation action may occur. To enforce these expectations, we use the Service Level rule type.



Escalation causes overdue assignments to become visible to users and managers so that they can be processed sooner rather than later. The urgency property determines the order that assignments for that work item appear on the worklist. Escalation recalculates the assignment's or work item's urgency to reflect its age, impending due date, or explicit management inputs. Escalation can also increase the urgency of all assignments for the work item, route it to another operator, send an alert or email message to the current operator, notify the manager of the operator, cancel the work, or initiate other processing. PRPC provides standard escalation activities, such as SendEmailToManagerAtGoalTime. For example, a developer can set a goal of 30 minutes to call a customer back and a deadline of four hours. The time interval usually starts when the assignment is created, not when a user begins processing the assignment.

Adding an SLA to an Assignment in a Flow

A service level rule can be associated with an assignment in the assignment shape's Properties panel. In the Service Level field, select the name — the second key part — of a service level rule to associate the service level with this assignment. When a clock icon appears on the Assignment shape, it indicates that a service level is applied to that assignment.



Calculating Response Time for a Service Level

Once work is pending for a user, PRPC maintains a timer for the pending work. The timer begins, by default, when the assignment is ready. The timer can also be configured to begin at a specific time, or after a specified delay.

The system uses this timer to monitor the completion of the assignment against the goal and deadline times and determine the urgency of an assignment at any point in time.

Once the goal milestone has been reached, the system performs the processing specified for the interval — updating the urgency of the assignment and performing any escalation activity. Once the deadline milestone has been reached, the system performs the processing for that interval, and begins calculating the response time for the passed deadline interval.

Unlike goal and deadline milestones — which are based on the assignment's initial availability — the passed deadline interval is calculated from the deadline milestone. We also can configure the passed deadline interval to repeat for a specified number of cycles. At the end of each cycle, the system updates the urgency and performs the specified action.

Calculating Urgency

When an assignment reaches a goal, deadline, or passed deadline time interval and is still unprocessed, escalation occurs automatically. Escalation updates the urgency value, reflecting the age, impending due date, or other explicit management inputs for the assignment.

Urgency is a system-derived numeric value between 0 (least urgent) and 100 (most urgent) that defines the importance of promptly completing and resolving an assignment, and determines the prioritized order of the assignments on a worklist.

As the goal, deadline, and passed deadline milestones are reached, the system adds the value of the corresponding Goal or Deadline Urgency field specified in the General tab to the assignment's urgency, so the assignment will appear higher on a workbasket list or worklist.

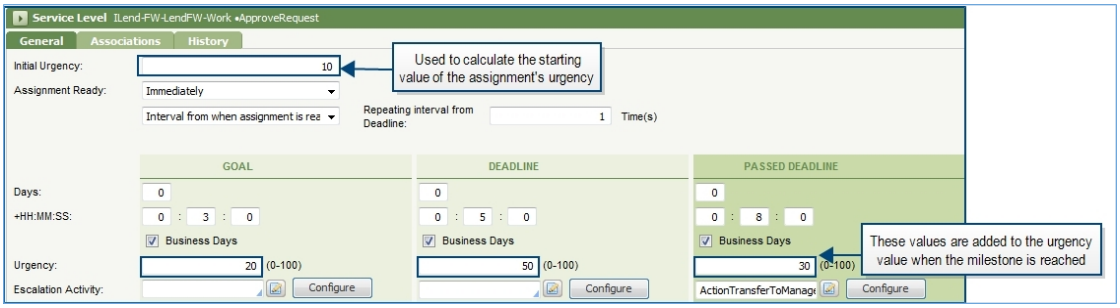
In the Case Manager user forms, by default, a value of 10 is considered normal priority. A red urgency clock in the row header indicates that the item has an urgency value equal to or greater than 60. A yellow urgency clock indicates an urgency value from 30 to 60. An urgency value less than 30 is indicated by a green urgency clock.

The urgency is the sum of the assignment's **pxUrgencyWork** property value plus an initial urgency value specified on the first tab of the Service Level rule form. Each time the interval elapses, the action is performed and urgency is calculated. Unlike response time, which elapses for each milestone independently, the urgency value for goal, deadline, and passed deadline is cumulative.

Consider an assignment with the following urgency values:

- An initial urgency of 10.
- A goal urgency of 10.
- A deadline urgency of 20.
- A passed deadline urgency of 10, with this interval repeating five cycles.

When the goal milestone is reached, PRPC updates the assignment's urgency to 20 — the initial value of 10 plus the goal urgency of 10. When the deadline milestone is reached, PRPC updates the urgency to 40. Each time the passed deadline interval repeats, PRPC increments the assignment's urgency by 10, until the interval completes 5 cycles, with a final urgency of 90.



Module 9: Automating Decisions

Lessons Covered:

- Creating a Decision Rule
- Decision Trees
- Creating a When Rule

Creating a Decision Rule

Objectives

- Identify decision rules
- Create a decision table
- Test decision rules
- Add a decision to the flow
- Delegate a decision rule

Things to Know

Decision Rules

We can automate business decisions by replacing manual steps and user actions with decision rules. A decision rule defines a computation or comparison; work item properties can be used as criteria in the calculation.

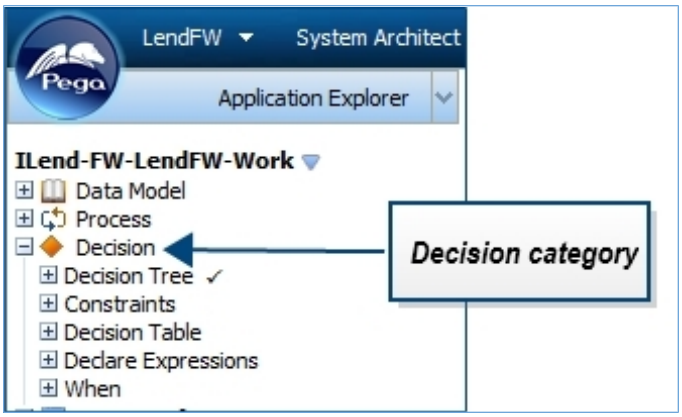
When the decision rule is run, the rule is evaluated. The resulting calculation returns a value (such as a string, number, etc.) that determines which path to take in the flow. The result is also stored in a property value.

Identifying Decision Rules

There are several common types of decision rules, including decision trees, decision tables, when conditions, and map values. Using the Application Explorer, we can find a list of all of the decision types in the Decision category.

- Decision tables - present business logic in the form of one or more if... then... else conditions that define a series of tests that are performed on property values. Each condition (and the criteria that comprise it) appears as a separate row in tabular form. When the decision table is run, each condition is tested, starting from the top of the table and progressing down through each row. Testing stops when the first condition evaluates to true, and the result for this condition is returned. Use a decision table when the criteria to be tested is consistent across all of the conditions.
- Decision tree - like a decision table, except that when a decision tree is run, testing starts from the top and progresses down through each branch that contains an applicable condition. Use a decision tree when the criteria to be tested is not consistent across all of the conditions. We will compare decision trees and decision tables shortly.
- When - (also called "when condition") defines a true-false test based on comparing on or more property values with constant values or with other property values.
- Map value - uses ranges for the input value or values to convert one or two input values, such as latitude and longitude numbers, into a calculated result value, such as a city name, available in a matrix.

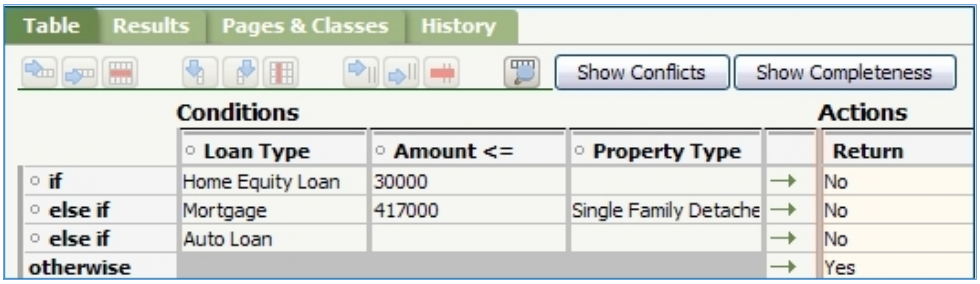
This lesson will focus on the decision table.



Creating a Decision Table

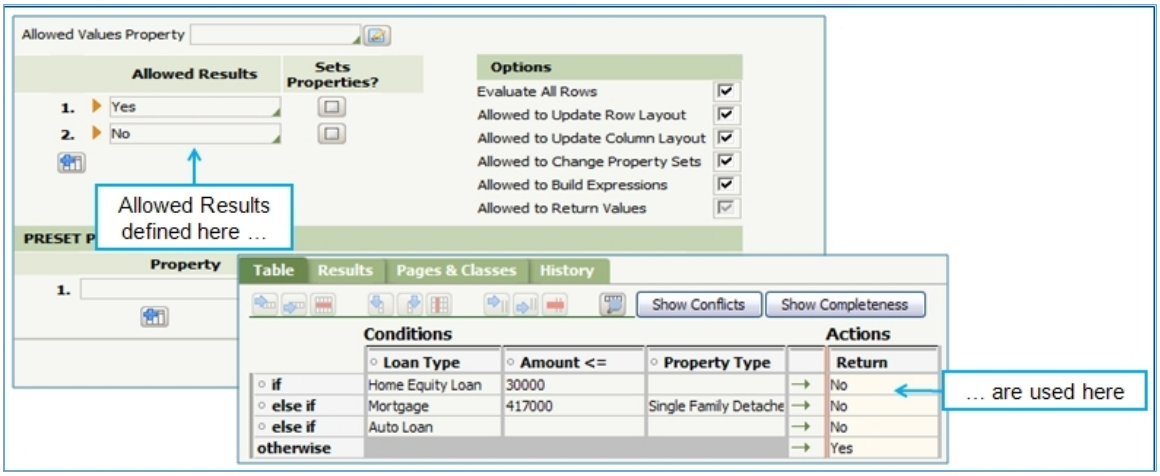
To define a decision table, we must specify the name of the class where it will reside. The decision rule must be defined in the same class as the properties it uses, or in a child class of those properties. We will cover classes in more detail in the Classes and Class Structure lesson. We must also specify the name of the decision table rule.

Next, we add one or more rows to the table. Each row contains the conditions to be tested and the intended resulting outcome. When all the test criteria for a row are evaluated as true the value from the **Actions** column is returned. For example, the name of the department, a status value, or Yes/No. If there is no test condition in a cell, then, by default, it returns a result of true. If no rows are true, it returns the value for the otherwise row.



Decision Table: Results Tab

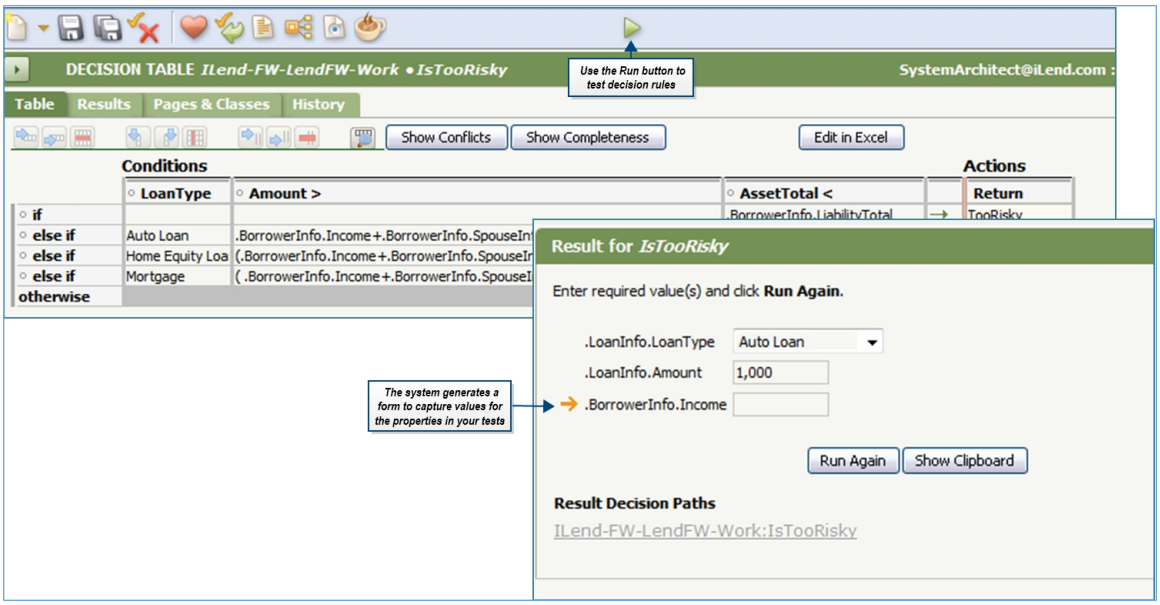
We can use Allowed Results on the Results tab to restrict the possible results that can be returned to a fixed list of non-null values.



Testing Decision Rules

We use the Run Rule feature to test a decision table or decision tree rule individually before testing it in the context of the application. To use the Run Rule we must:

1. Use the Run button
2. Provide values for each input
3. Examine the results.



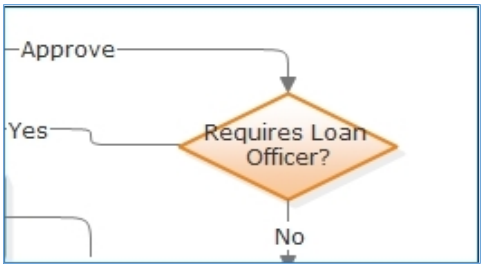
Bear in mind that PRPC will only test input values that have been explicitly specified, so it will not override calculated values during testing. This means that if the input criteria is a calculated value, the condition may not be fully tested.

Adding a Decision Shape to a Flow

To add a decision to a process we insert a Decision shape in a flow. The Decision shape calls the specified decision rule.

Decision Shape

The label of each connector emanating from this shape is based on the allowed results specified in the Results tab.



We can reference a decision table by a:

- Decision shape in a flow rule
- Declare expression rule
- Collection rule
- Decision tree, decision table, map values, or from an activity.

We can also reference a decision table through a decision tree in an activity that can be evaluated using the Property-Map-DecisionTable method.

Delegating a Decision Rule

After initial development and testing of a decision table has been completed, selected decision table rules which might require frequent updates can be delegated to business managers who can make updates directly, allowing them to build for

change. The Decision tab of the decision tree form and the Table tab of the Decision Table form provides managers with access to the fields updated most often, subject to any restrictions entered on the Results tab. Depending on access roles and privileges, the manager can update the entire rule form or only fields in the leftmost tab of the rule form.

A rule can be delegated to a single user, to multiple users, or to all users who are associated with a specific access group. To delegate a rule, a system architect makes that rule a favorite for their own use, another individual or an access group. The individuals can view a list of their delegated rules when they log on to their portal.

Bear in mind that delegated rules must be segregated in an unlocked RuleSet and version so that managers can change them. As a convenience, do not require check-out for this RuleSet.

Decision Trees

Objectives

At the end of this lesson, you will be able to:

- Create a decision tree
- Apply conditions to a decision tree
- Describe how the Results tab is used
- Know the best uses of decision trees and decision tables

Things to Know

What is a Decision Tree

We use a decision tree to calculate a value based on one or more rows containing if/then logic. The decision tree can return a result that is not limited to true or false, such as a property value. We can create a row that contains multiple criteria that defines a series of tests performed on property values, computations, and comparisons in order to automate a decision. For example suppose we want to calculate a 5 PM order cutoff time on business days, but a 2 PM cutoff time on Fridays that fall before a Monday business holiday.

The decision tree format is easier to compose and understand than a series of related when conditions. For example, since a decision tree can call other decision rules, a large decision tree can be organized into several, more manageable branches.

Decision Tab

The Decision tab is where the functions used when testing are recorded as if/then logic in the three-column array known as the comparison, action and next value columns. When the decision tree is run, the system evaluates the if

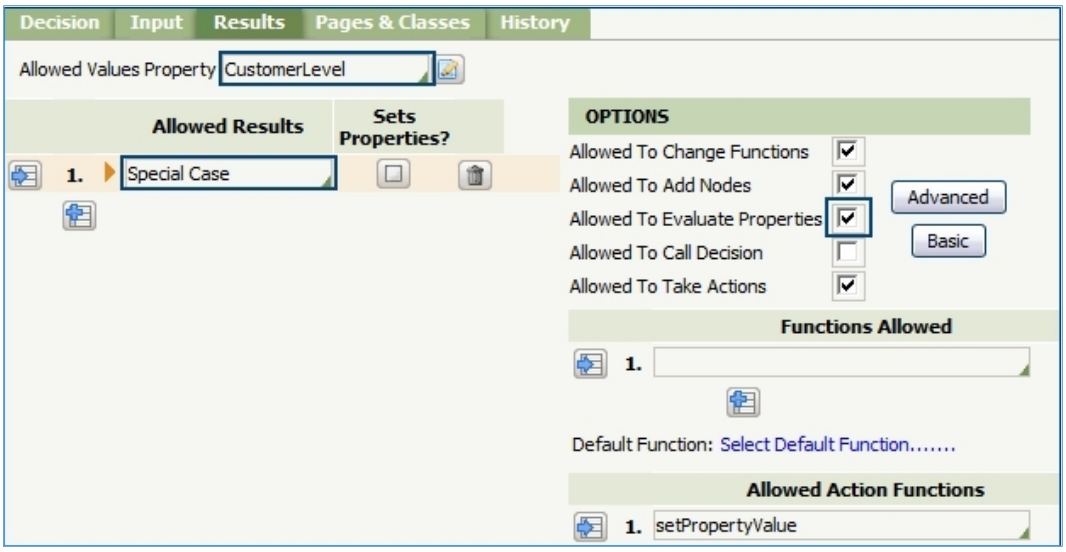
portion of the array, starting at the top row, and continues until it reaches a Return statement. If the entire tree is processed without reaching a Return statement, the Otherwise value is returned. The Decision tab offers various formats and choices, depending on the settings found on the Results tab.

Results Tab

The Results tab is an optional tab that we can use to restrict the possible test results to a list of acceptable, non-null constant values. If expressions or property references are entered as result values, the system evaluates them each time it evaluates the decision tree.

If no options are selected on the Results tab, the decision tree returns results based on the values on the Decision tab (if/then statements, otherwise, and continue). If the Allowed to Evaluate Properties checkbox is selected, we can specify a property in the Allowed Values Property field to evaluate. This property must be in the Applies To class (or in the inheritance path) that offers a list of values that are allowed results for this decision tree. For example, we can specify the CustomerLevel property in the Allowed Values Property. This property contains a local list of values, such as gold, silver and platinum.

We can also specify a list of allowed results in the Allowed Results field and associate each with a set of property value changes that occur when this result is computed at runtime. If the Allowed Values Property field is not blank, values you list in the Allowed Results field supplement values provided by that property. For example, if the CustomerLevel evaluates to Gold, we can use the Allowed Result Special Case to list additional conditions, like lower interest rates.



When creating a basic decision tree, complete the Results tab before the Decision tab to restrict the results to one of several constant values.

Returning a Valid Result

We can define a list of property updates that are applied before the system processes information on the Decision tab. At runtime, the system updates property values in the order listed.

Which Decision Rule Should We Use?

Use the table below to help you figure out which rule to use.

Decision Table	Decision Tree
The tests are often based on the same set of	The tests tend to be for different properties

Decision Table	Decision Tree
properties	
Logic is easy to follow, and can be exported to MS Excel	Flexible format, good for complex if/then and nested conditions
Preferred by Business Analysts	Familiar to System Architects with software development experience

Creating a When Rule

Objectives

At the end of this lesson, you will be able to:

- Create a when rule
- Manage when rule logic
- Use a when rule in a flow
- Describe other uses for when rules

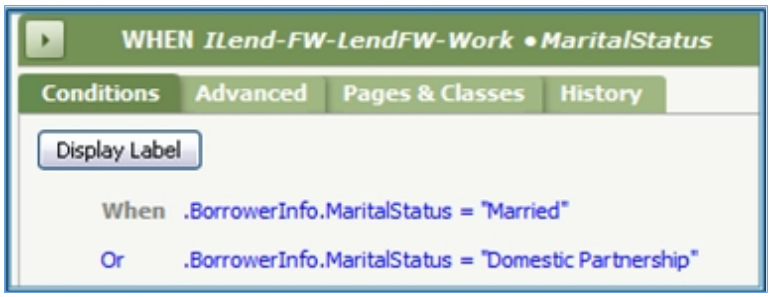
Things to Know

What is a When Rule?

The when rule is used to evaluate the relationship between property values with constant values or with other property values for one or more Boolean logical statements, and returns a value of true or false. It is similar to the *if* portion of the *if-then* construct in many programming languages.

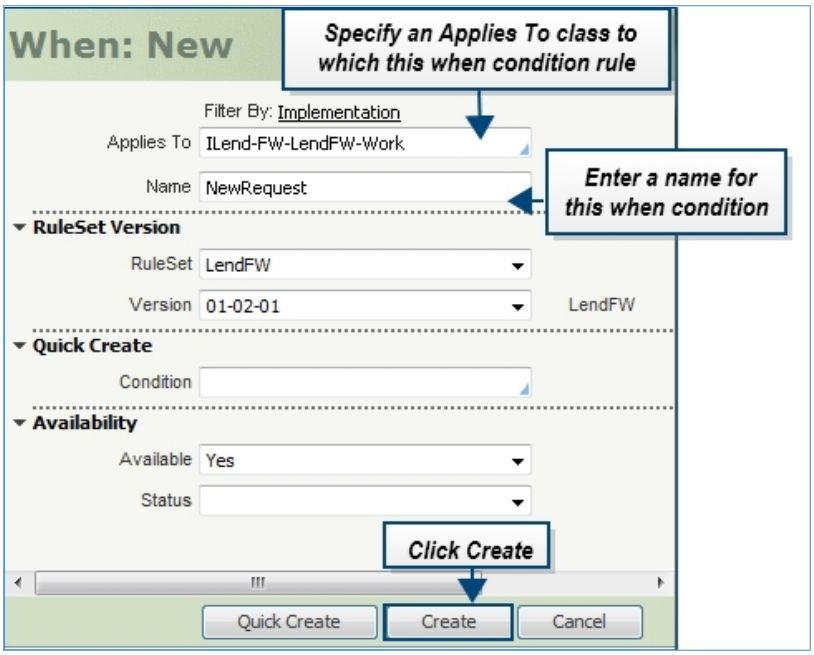
A when rule can be used by other rules or shapes. For example, a flow rule can use a when rule to direct work item processing. We can also use a when rule if we want to test if:

- A grand total that is greater than zero
- A date value is in the future

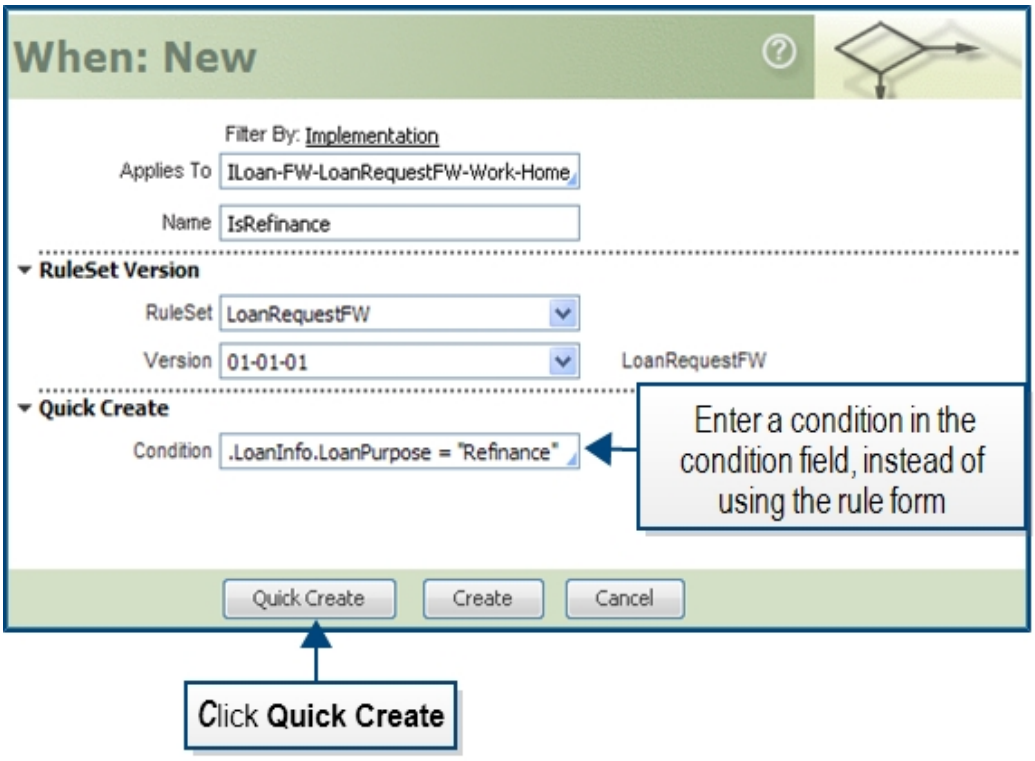


Creating a When Rule

We can create a when rule with one or more conditions. We create a when condition rule by selecting When from the *Decision* category.

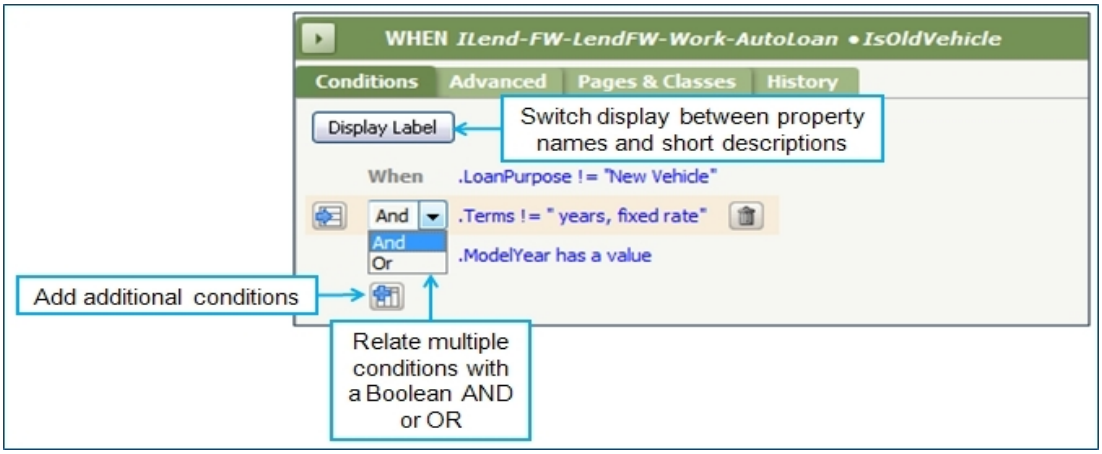


If our condition is a simple Boolean expression, we can enter it on the New Instance of a Rule dialog and click Quick Create to create our when rule without using the rule form.



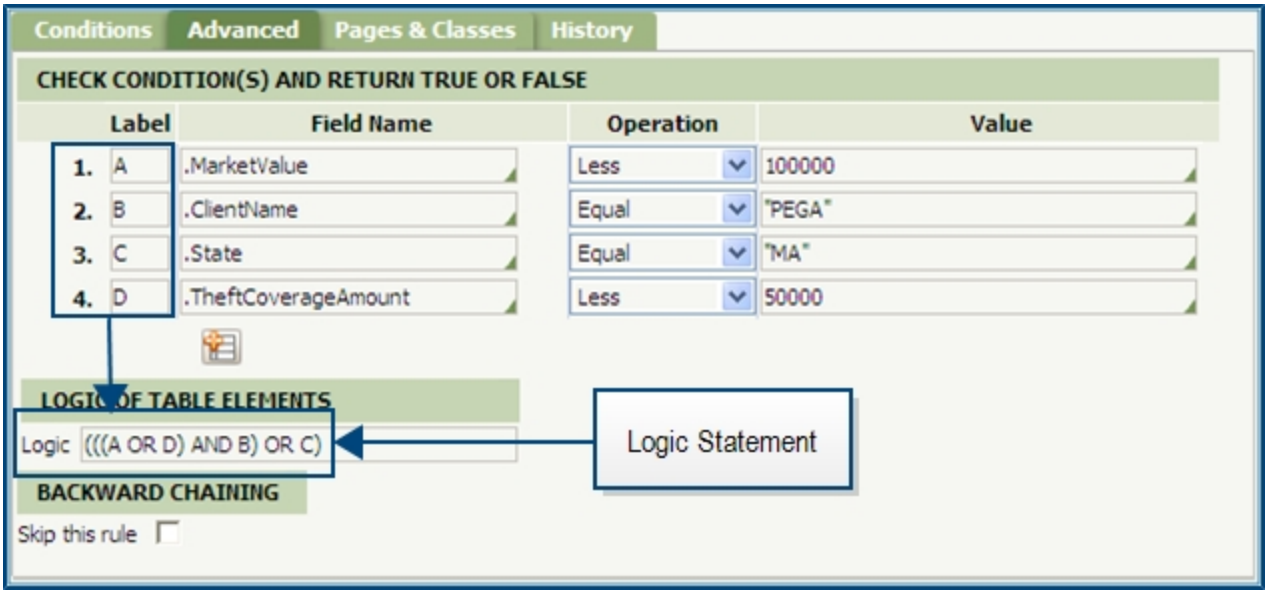
Specifying a Condition on a When Rule

We use the Conditions tab to enter test conditions that can be expressed as a single Boolean condition, or to relate multiple test conditions using a Boolean AND or OR. Each row includes a single comparison or other condition that evaluates to True or False. The when condition evaluates to True only if all rows evaluate to true. The order of the rows does not affect the outcome of the when condition evaluation, but may affect performance.



Managing When Rule Logic

The Advanced tab of the when rule is used to specify more complex relationships, for example when combining conditions using AND, OR, and NOT operators. Parentheses must be used to enforce the evaluation order of multiple conditions. Specify comparison details in the array and enter a logic statement that combines all the comparisons. Once the rule form has been saved, the system-generated Java code for this rule can be reviewed and the rule can be tested. If the Advanced tab is used to build the conditions, the Conditions tab will not be available and thus cannot be used to edit this when rule.

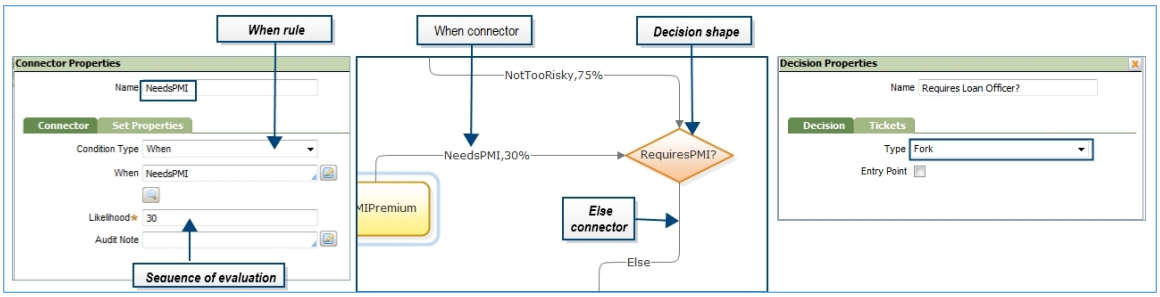


Using a When Rule in a Flow

A decision shape can be configured to act as a fork. When this happens, the decision logic is not associated with the decision shape; rather, the logic is associated with each connector. The connectors that emanate from a fork reference when rules.

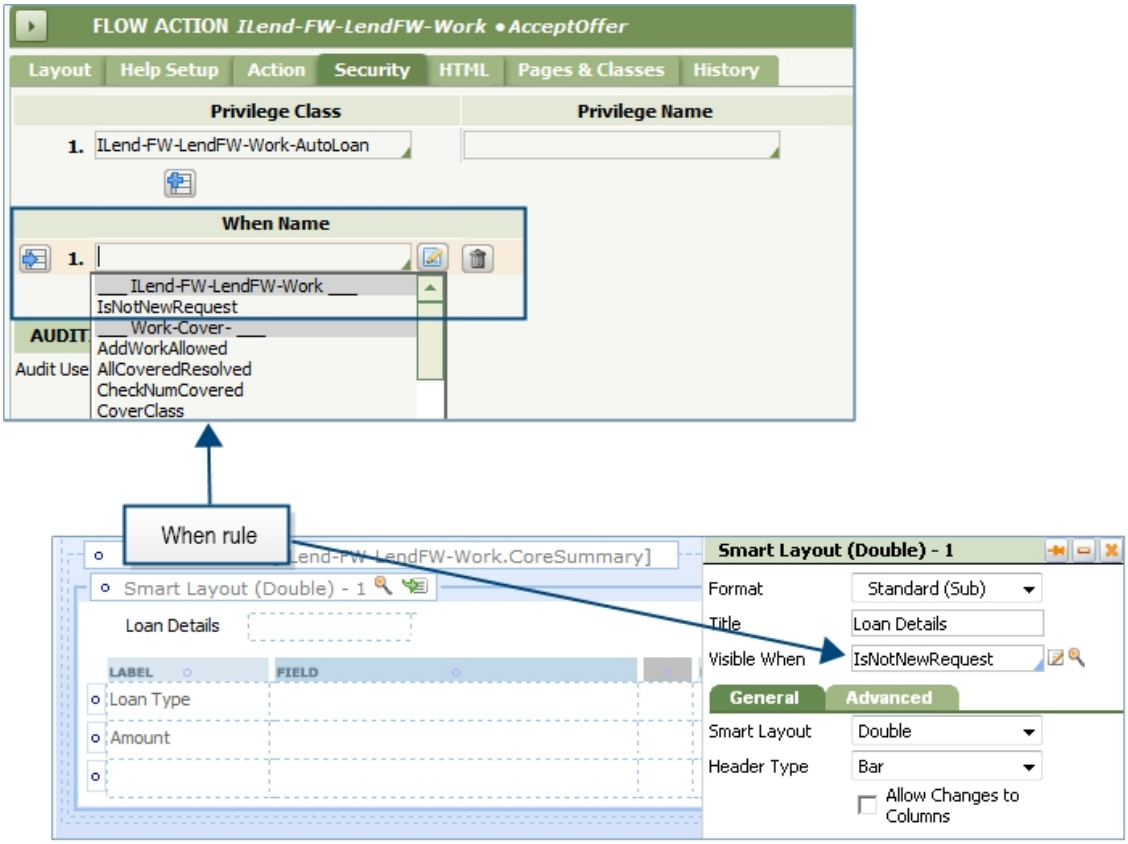
Each fork may have multiple connectors (each with its own when rule) emanating from it. As a best practice, always include an Else connector with every fork. This ensures that PRPC has a valid path from the fork to another shape.

The connectors are evaluated in descending order of likelihood, and evaluation stops with the first true result. The Else connector is used when none of the when rules evaluates to true.



Other Uses of When Rules

We can use a when rule to conditionally allow users to perform an action. For example, a UI rule can use a when rule to conditionally display a field and its content.



Module 10: Single-Variable Circumstancing

Lessons Covered:

- Circumstancing Rules
- Single-Variable Circumstancing

Circumstancing Rules

Objectives

At the end of this lesson, you will be able to:

- Explain rule circumstancing
- Describe when and why to use circumstanced rules
- Discuss the types of rule circumstancing available

Things To Remember

What is Circumstancing?

A circumstance is an optional qualification available for all rules. Using a circumstance allows our application to support multiple variants of a rule. For example, variations of an ordinary rule instance can be created for different customer status levels or for different geographic locations. Rule Resolution uses circumstancing to find the best rule in any situation. The circumstanced rule is executed during rule resolution if the value of the circumstance property on the rule matches that property's value on the clipboard at runtime.

Unlike other methods of controlling rule access — such as rule versioning or adding and removing RuleSets and/or classes from an application — circumstancing allows PRPC to pick the correct version of the rule when the rule is used, rather than during the development phase. This flexibility allows us to create more-powerful and more-robust applications.

For example, assume that we have different pricing levels for our customers. We begin by defining a base pricing rule for all customers. Then we qualify the base rule by creating circumstanced rules for our customers at different buying levels. The property .CustomerType is part of the customer order and has the values of either "Silver" or "Gold". In this example, our customer purchased a $100 item. Using the property and values, we create circumstance-qualified instances of the base rules as shown below:

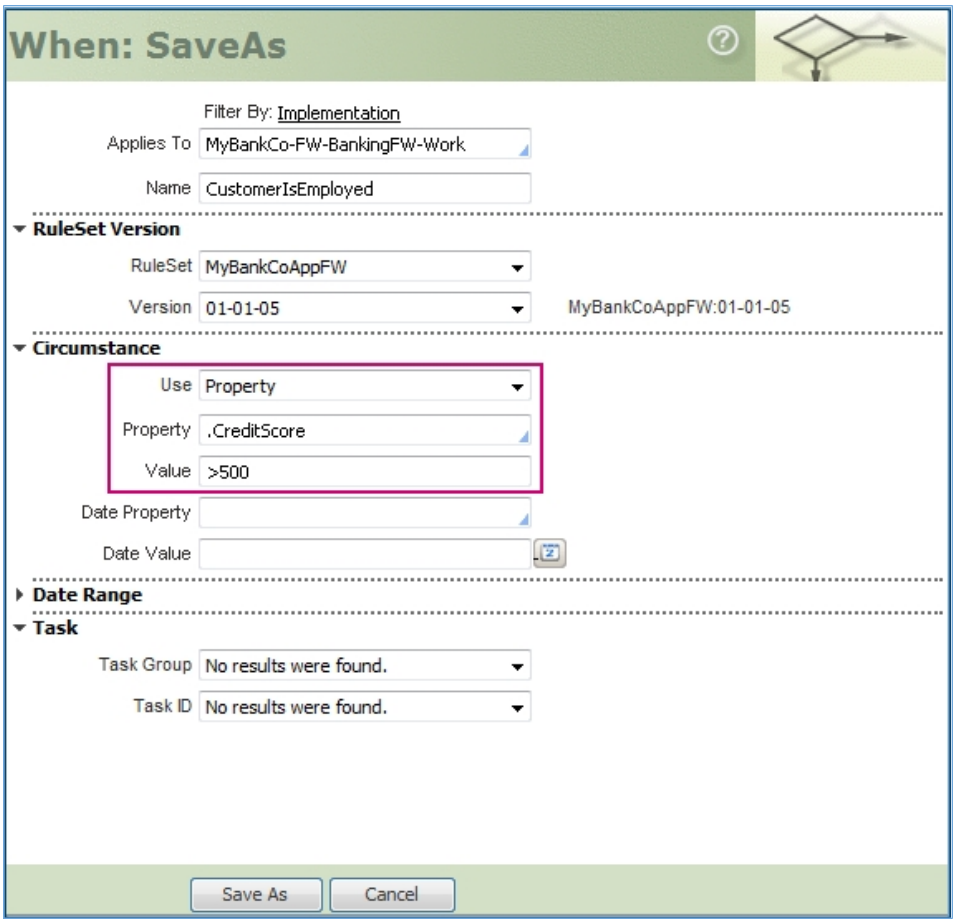
- BasePrice rule — if .CustomerType=(none), then the price is $100
- BasePrice circumstance 1 — if .CustomerType = “Gold”, then the price is $100, minus a 25% discount
- BasePrice circumstance 2 — if .CustomerType =“Silver”, then the price is $100, minus a 10% discount

When the system processes the order, the value of that property dictates which rule is run and thereby determines the discount (if any) the customer receives.

Types of Circumstancing in PRPC

Single Variable

After creating a rule, we can define qualifiers which may be used to create additional versions of our rule for specific situations. With single-variable circumstancing, we provide a single condition for the rule. To use this type of circumstancing, copy the original rule using Save As. In the Save As dialog, select the property you wish to use as a condition, and specify the value for the condition.



Multivariate

With multivariate circumstancing, we can circumstance a rule using multiple properties and/or property value ranges. We do this by building a circumstance template and a circumstance definition rule to hold our definitions. These definitions can be used by more than one rule. However, if a circumstance changes, only the multivariate rule must be updated. Multivariate circumstancing is a Senior System Architect responsibility.

Single-Variable Circumstancing

Objectives

At the end of this lesson, you will be able to:

- Circumstance rules according to a property value
- Circumstance rules according to a date value
- Qualify rules with a date range
- Identify qualified rules

Things to Know

Circumstancing allows us to create multiple versions of a rule that can be used as the situation dictates. In this lesson, we'll look at **single-variable circumstancing** and **time-qualified** rules.

Qualifying Rules

The simplest form of circumstancing is single-variable circumstancing, which allows us to call different rule variants based on a single value. PRPC provides two options for single-variable circumstancing: **property** and **date**.

- **Property** — the rule is circumstanced according to a *property value*. If the specified property has been set to the specified value, PRPC calls the circumstanced version.
- **Date** — the rule is circumstanced according to a property name and a comparison *date value*. At runtime, the system compares the value of the property on the clipboard to the comparison value, and chooses the date circumstance rule (if any) with a Date Value that is closest, but not later than, the property value. We sometimes call this *as-of processing*.

Similar to date circumstancing, PRPC also allows rules to be time-qualified. When a rule is time-qualified, the rule is qualified by a *date range*. If the current system time falls within this range, PRPC calls the circumstanced version.

How Do Qualified Rules Work?

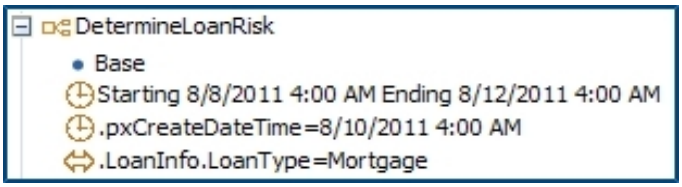
To create a qualified rule, we must first create a **base rule**. This is the rule that PRPC uses if PRPC cannot identify a more-appropriate, qualified variant.

Once we have created the base rule, the next step is to create the **qualified rule**. To do this, we create a new instance of the same rule, by clicking Save As and completing either the **Circumstance** portion (for a circumstanced rule) or the **Date Range** portion (for a time-qualified rule) of the dialog. Qualified versions of the rule must be saved to either the same class as the base rule, or one of its child classes.

Once a qualified version of a rule exists, PRPC uses the following symbols to differentiate between a base rule and a qualified rule.

-  — indicates the **base rule**, the version PRPC executes if no more appropriate circumstanced version can be found. No indicator appears in the Application Explorer.
-  — indicates the **property-circumstanced** version of the rule. PRPC displays this symbol and the circumstance in both the Application Explorer and the rule header.
-  — indicates the **date-circumstanced** or **time-qualified** version of the rule. PRPC displays this symbol and the date circumstance or date range in the Application Explorer and the rule header.

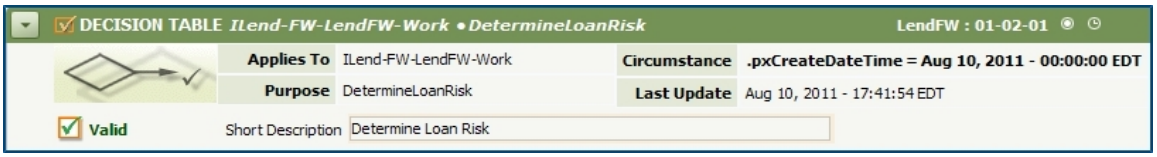
In the Application Explorer, PRPC collects all the versions of the rule — the base rule and all of the qualified rules — under the rule name as an expandable entry. Expanding the entry exposes all of the variants. The base rule appears with the label "Base", and each qualified variant appears with its qualifying criteria as the label.



In the previous example, the variants are listed in the following order:

1. Base (unqualified) rule.
2. Time-qualified rule. PRPC indicates time-qualified rules with the labels **Starting** and/or **Ending**.
3. Date-circumstanced rule. PRPC indicates date-circumstanced rules with the date property and the comparison value.
4. Property-circumstanced rule.

In the rule header, PRPC displays the circumstance of the qualified rule.



In addition, the qualifying information is also available by hovering over the qualified symbol in the rule header.

Processing Qualified Rules

When PRPC calls the rule, it assembles all of the qualified variants and the base rule, and determines which is the most appropriate for the situation.

For a time-qualified rule, PRPC selects the rule to run based on the following tests:

1. PRPC examines the End dates on each candidate time-qualified rule, and chooses the rule or rules that has the nearest End date, discarding the others.
2. If only one candidate remains, that rule is the result of this phase of rule resolution processing.
3. If two or more candidates remain, the one with the most recent Start date is selected.
4. If no qualified version remains, PRPC selects the base rule.

For a circumstanced rule, PRPC selects the rule to run based on the following tests:

1. If a property circumstance is specified, the property on the clipboard must match exactly the circumstance property value.
2. If a date circumstance is specified, the Date Value in the rule must not be later than the clipboard property date value, and closer than any other qualified rule with the same Date Property.
3. If both a property circumstance and a date circumstance are specified, the rule is selected only if **both** test

conditions are met.

4. If no qualified version remains, PRPC selects the base rule.

Module 11: Using Declarative Rules

Lessons Covered:

- Declarative Rules
- Declare Expressions
- Declare Pages
- Constraints

Declarative Rules

Objectives

At the end of this lesson, you will be able to:

- Define declarative evaluation
- Name and discuss the Declare rule types
- Describe processing based on change
- Explain and use declarative network analysis

Things to Know

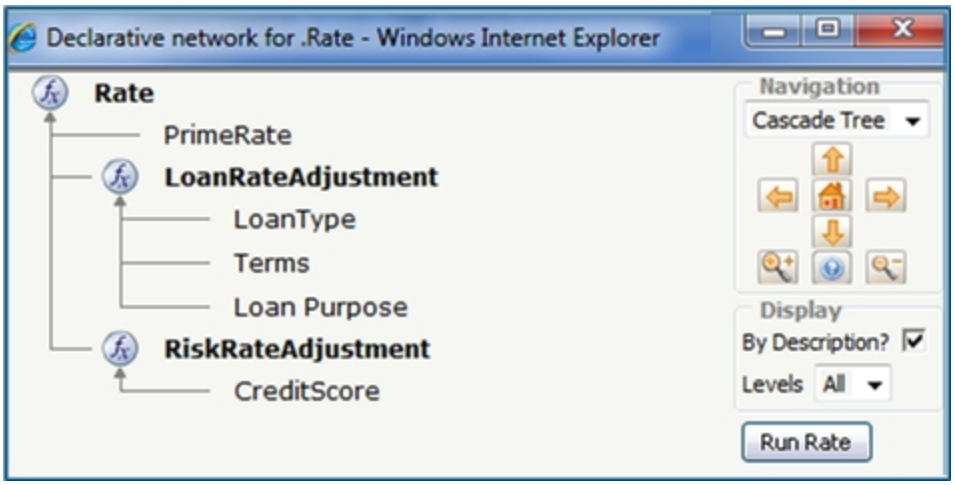
Declarative Processing

Declarative processing simplifies your application and reduces the number of activities you need to create. A declarative rule describes a computational relationship among property values that is expected to be valid "always" or "often" or "as needed." In practice, declarative rules are evaluated automatically when needed to satisfy certain conditions. These rule types differ in processing to offer you specific meanings of "often" or "as needed."

The primary benefit of declarative processing is that the system, rather than the developer, controls when computations are processed. Relationships between property values can be expressed directly, without the need to create activities and design the application to run the activities often.

Declare Rule Types

Declarative rules provide a way to process property values automatically. They are run when a value is set or changes and do not need to be called explicitly or run in a sequential fashion. Declarative rules allow relationships between properties to be expressed directly, without the need to create and call activities. They model the relationship visually, and let the system generate the programming. Using this type of rule speeds development and simplifies maintenance.



The declarative processing facilities support the operation of six rule types:

- **Constraints** (*Rule-Declare-Constraints* rule type) — define and enforce comparison relationships among property values. Constraints can provide an automatic form of property validation every time the property's value is "touched", in addition to the validation provided by the property definition or validation rules. The system evaluates

constraints automatically each time a property identified in a constraint is changed.

- **Declare Expression** (*Rule-Declare-Expressions* rule type) — define automatic computations of property values based on expressions. This type of rule is discussed in more detail in the *Declare Expressions* lesson.
- **Declare Index** (Rule-Declare-Index rule type) — define criteria under which PRPC automatically maintains index instances for faster access. An index can improve search and reporting access for properties that cannot be exposed as database columns because they are embedded within an aggregate property. Index instances are sometimes called alternate keys or secondary keys. The system saves indexes as instances of concrete classes derived from the Index- base class.
- **Declare OnChange** (*Rule-Declare-OnChange* rule type) — run an activity automatically at activity step boundaries when the value of a specified property changes. This capability provides a form of automatic forward chaining. Declare OnChange can force all processing on a work object to be suspended pending an independent review of the situation.
- **Declare Trigger** (*Rule-Declare-Trigger* rule types) — cause an activity to run when instances of a specific class are created, updated, or deleted in the database. This implements a form of forward chaining. For each Declare Trigger, PRPC monitors database operations for objects of the Applies To class (and concrete classes derived from that class). During database commit processing, forward chaining processing may trigger — start execution of — the activity identified in this rule.
- **Declare Pages** (*Rule-Declare-Pages* rule type) — create a clipboard page when a declare page is referenced. Declarative pages can improve performance and reduce memory requirements when all or many requestors in an application need to access static information or slowly changing information. This type of rule is discussed in more detail in the Declare Pages lesson.

Processing Based on Change

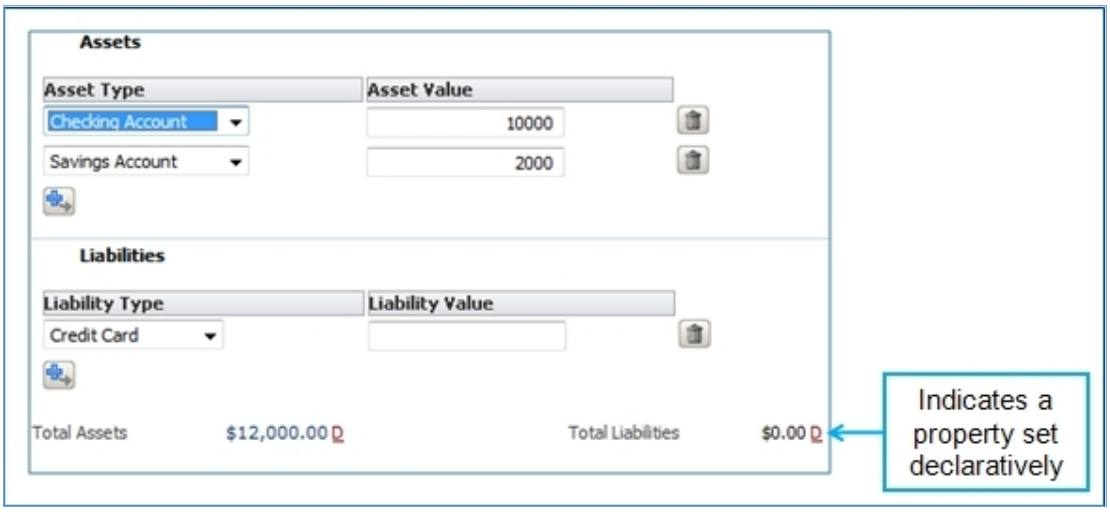
You can establish required relationships among properties in a Declare Expressions, Constraints, Declare OnChange or Declare Trigger rule. When the value of a property is involved in any of these declarative rules, the system automatically checks an internal dependency network for other values that are affected and performs other processing as determined by the network. This technique is known as **forward chaining**. (For an explanation of forward chaining, refer to the *Declare Expressions* lesson.)

After the following types of events the system reevaluates most declarative rules:

- When users submit an input form, evaluation occurs at the conclusion of input processing.
- As an activity runs, evaluation occurs during each step, after the method completes but before evaluation of a transition in that step.
- During flow execution, as control advances from one task (one shape on the flow diagram) to another.
- At the conclusion of a flow transition, when the work object advances from one shape to another and within connectors (if a relevant property is set).

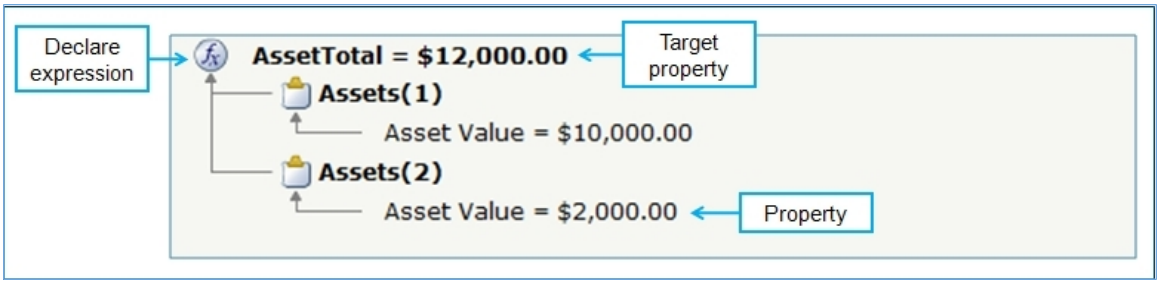
Using the Declarative Rules Inspector

We can quickly see what properties are set declaratively by using the Declarative Rules Inspector. From the Quick Launch toolbar, select **Run > Rules Inspector > Declarative Rules**. Start the process you want to inspect. A **D** appears next to a field to indicate the use of a declarative rule.



Declarative Network Analysis

The declarative dependency network is an internal data structure that supports automatic recalculation of certain property values based on changes to other property values. When the Declarative Rules Inspector is running, we can click the **D** next to a declaratively-set property value to open the Dependency Network display. In the Dependency Network display, we can select properties and enter values to test the dependency network.



The Declarative Network Analysis gadget — available on the Declarative Network landing page — displays a list of declarative networks for each class in your application. Each list contains:

- The declarative network name and level (top or intermediate). To open the Dependency Network display, click a declarative network name.
- The declare expression used to calculate the declarative network. Click the Declare Expression icon to open the declare expression rule. To open the target property, click the property name.
- And the method of calculation for the target property.

To view all declarative networks, select the Include Pega Application option.

Declare Expressions

Objectives

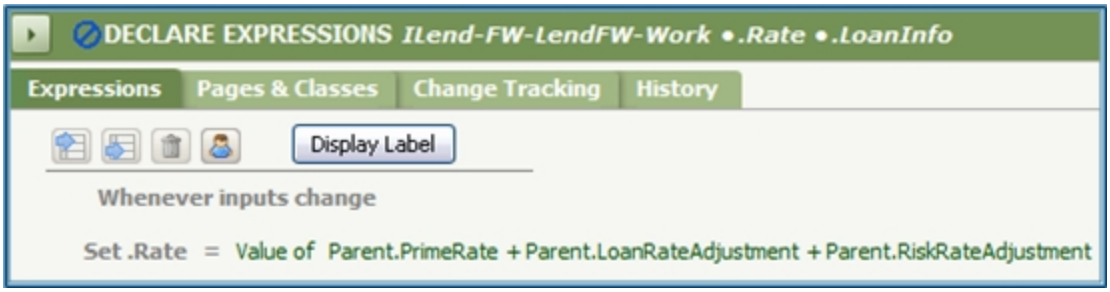
At the end of this lesson, you will be able to:

- Define and discuss declare expressions
- Create a declare expressions rule
- Discuss dynamic expression calculation

Things to Know

What are Declare Expressions Rules?

Declare Expressions rules, listed in the *Decision* category of the Application Explorer, **define automatic computations of property values based on expressions**. Don't confuse Declare Expression rules with simple expressions. Expressions — a syntax that includes constants, function calls, operators, and property references — are used in many situations in addition to Declare Expression rules.



Declare Expressions rules are typically evaluated using a technique known as **forward chaining**. We can also use a technique known as **backward chaining** to perform a computation when the value of an input or parameter property is not available.

Forward Chaining

Forward chaining is a technique PRPC uses to compute target property values automatically each time any of the other input values in the expression change, or when a value for this property is accessed or changed. For example, when we're collecting information about a customer's assets, a declare expression can automatically update the total sum of the customer's assets when an asset amount is added or updated. Forward chaining is the default technique used in declare expressions rules.

Backward Chaining

With backward chaining, PRPC *only* executes the expression **when the target property is used**. This can include more complex chaining, such as when values in expressions depend on the result of other declare expressions rules. The use of backward chaining provides gains in efficiency over the default forward chaining method, by delaying computation of an expression until the target property is used.

To enable backward chaining on a declare expression, select one of the following computation methods on the Change Tracking tab of the Declare Expressions rule:

- When used, if no value present.
- When used, if property is missing.
- Whenever used.

We can also trigger a backward chain of calculations using a process called **goal seeking**, regardless of the computation method used for the declare expression. In this situation, PRPC searches for all of the input values required for a calculation. PRPC reports on any missing input value(s), and we can configure our application to either determine the missing values programmatically, or query the user for the missing values. We can implement goal seeking in one of two ways:

1. By using an activity that contains the Property-Seek-Value method. This method accesses the computational relationships among properties in a Declare Expression rule and identifies source values needed to compute a missing property value.
2. By using the standard flow action VerifyProperty in your flow and specifying the target property to evaluate.

Dynamic Expression Calculation

Dynamic expression calculation by default updates a calculated value (the target property) when any value in its expression changes. To automatically display updated property values, select the Enable Expression Calculation checkbox on the HTML tab of a harness or flow action.

Reference Material

PRKB-24262 — How to trigger backward chaining in flows (goal seek)

Declared Pages

Objectives

At the end of this lesson, you will be able to:

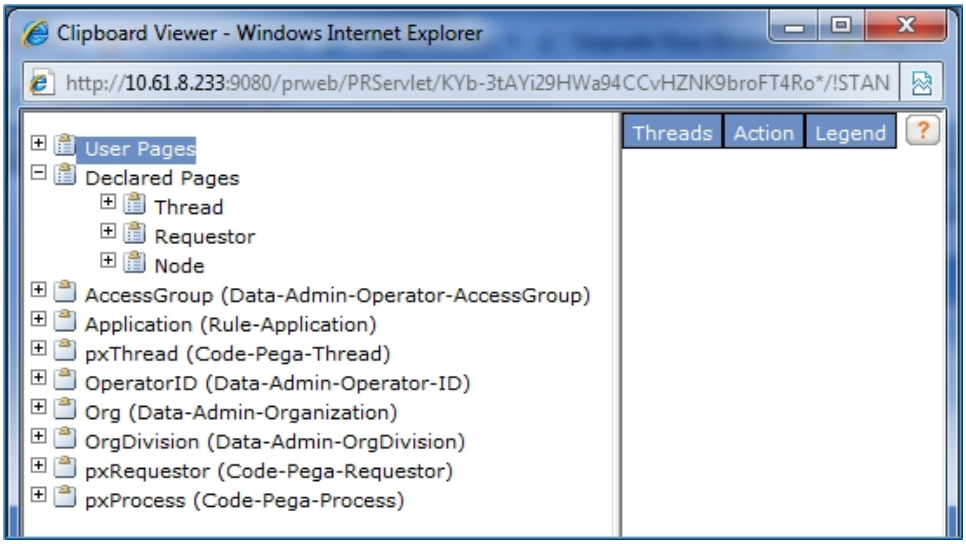
- Define a declared page
- Explain when you should use a declared page
- Discuss expiring and refreshing declared pages

Things to Know

What is a Declared Page?

A declared page is a clipboard page created by execution of a declare pages rule. Three types of declared pages can be written to the clipboard.

- **Thread** – enables data sharing within a single requestor thread.
- **Requestor** – enables data sharing between all If a requestor's threads.
- **Node** – enables data sharing between requestors.



The name of a declared page starts with the prefix Declare_. Such pages are created and updated only through the operation of activities or report definitions identified in the corresponding Declare Pages rule. These pages are visible to requestors as read-only content.

Node-level Declare Pages define the contents of a clipboard page that is to be available in read-only mode to multiple requestors in your application. For example, a page may contain daily prices for a set of commodities, bonds, or currencies, an organization chart, or quantity-on-hand inventory data extracted hourly (rather than in real time on-demand) from an external system. These pages can be "global", available to all requestors on a node.

Declared pages are not created at system startup; they are created when first accessed. Declare Pages rules are not called by any rule in applications but instead are executed as a result of declarative network processing. Rules access the declared pages created on the clipboard as a result of Declare Pages execution. Referencing a declared page in a rule triggers its creation and availability on the clipboard.

Ordinarily, declarative pages do not persist; they are not saved to the PegaRULES database but are reconstructed when next accessed and the previous version is expired.

When Should We Use a Declared Page?

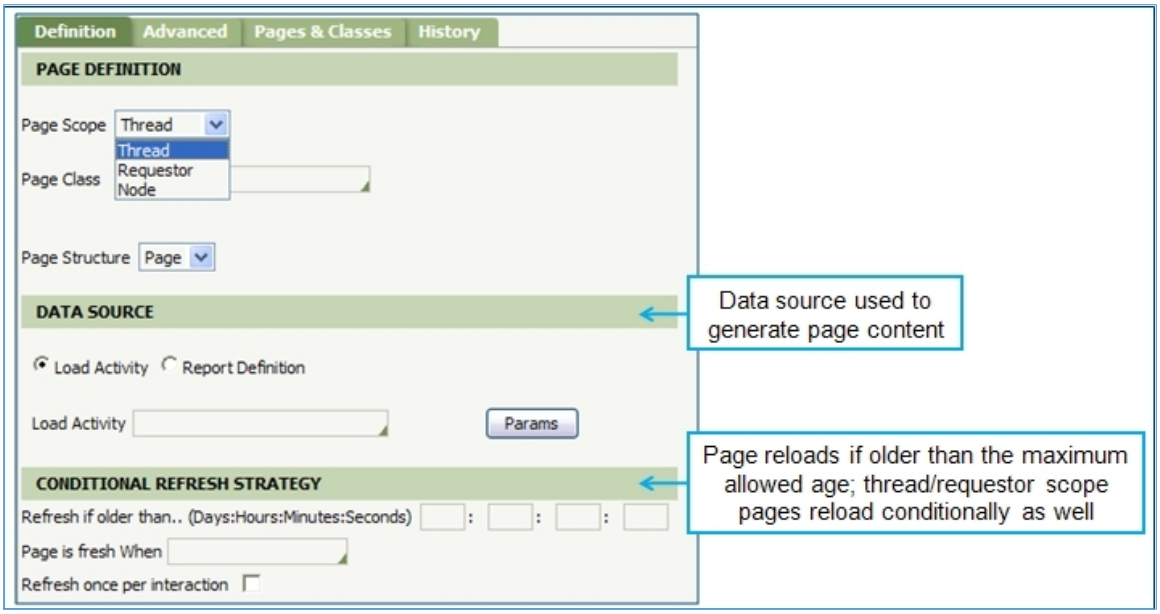
We may use declared pages when we want to improve performance and reduce memory requirements when processing requests for the same static information or slowly changing information. We typically use declare pages when the data is shared across Pega applications, either for one requestor or all requestors on a node.

In other situations, data values are not static but may change infrequently; PRPC automatically checks the declared page contents — either against a pre-set time limit or by using a when condition — before each property access to see whether a fresh recomputation is needed. For example, a page may list the part numbers or SKU numbers of items that are out-of-stock, extracted from an inventory control system. Recomputation is needed only when the out-of-stock condition begins or ends, not each time the inventory changes.

We can also use a declared page to access to a list of results from a database using a report definition. This allows us to pull results onto the clipboard by simply referencing the declared page, and without writing or invoking any activities.

Expiring and Refreshing Declared Pages

Declared pages can conditionally "expire", which causes PRPC to refreshes the next time they are referenced. The "Refresh if older than", "Page is fresh When", and "Refresh once per interaction" settings control when a declare page expires.



For example, we can define a declarative page that expires after 60 minutes. If the first-ever access to the page occurs at 9:15 AM, the system computes the page and makes it available in read-only mode. Other accesses occur at 9:30 AM and 10:05 AM and are served by that page with no recomputation occurring. At 10:15 AM, the page is considered expired, although the system does not detect this until the next attempt to access the page.

Constraints

Objectives

At the end of this lesson, you will be able to:

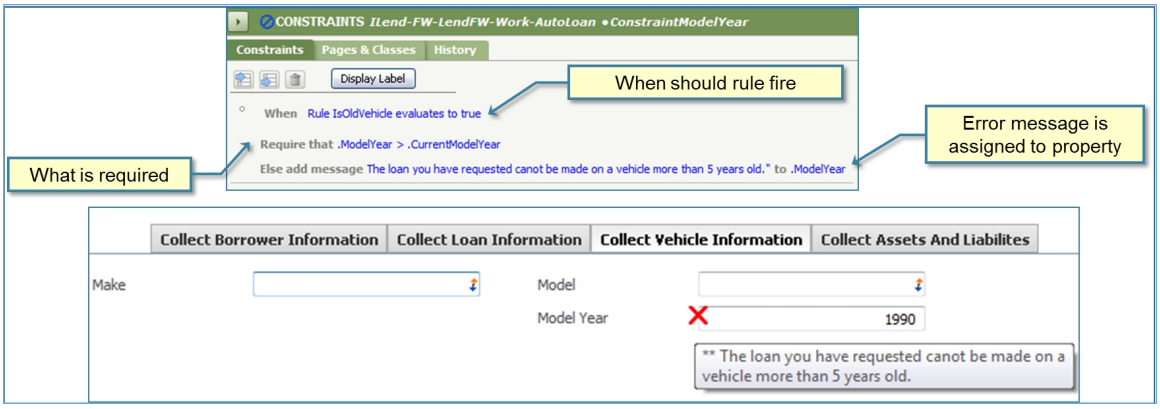
- Explain constraints
- Describe when to use constraints
- Create a constraints rule
- Identify alternatives to constraints

Things to Know

What is a Constraint?

A constraint is a comparative relationship between property values (or between a property value and a constant or expression) that is expected to be true at all times.

For example, our application may require that an interest rate entered by a user be greater than or equal to 4.25 percent and less than or equal to 9.00 percent. Alternately, we might want to make sure that an actual expense for a particular month is never greater than the budgeted expense for that month.



When to Use Constraints

Validation occurs when a property's value changes. In addition to the "global" validation provided by the property rule, a constraint rule can provide an automatic form of property validation every time the property's value changes, using a technique known as **forward chaining**. (For an explanation of forward chaining, see the *Declare Expressions* lesson.)

Constraints rules are evaluated when the value of any property mentioned in the rule changes. If a constraint is not met, the page containing the property is marked with a message, making the page invalid and the user is prevented from proceeding until they satisfy the constraint by correcting their input.

When a constraint fails, the clipboard page containing the property is marked as invalid and ordinarily the page is prevented from being saved to the database. The invalid value remains on the clipboard but the user is not automatically notified of the invalid value. The application's user interface determines whether, when, and how a user learns about this validation failure and when they are prompted to correct the input. We can define a set of constraints on a property once, and it will always fire on any instance of that work type.

When using a constraints rule, we need to remember the following points **to avoid trapping the end user with an irresolvable error**:

1. Constraints are evaluated declaratively by PRPC. We do not call them explicitly within our application.
2. A constraint violation must be resolved before the end user can submit the current form.

Consider the following situation:

- A bank prohibits loans longer than three years on vehicles more than five years old. We create a constraint to enforce this prohibition.
- A loan applicant requests a used-car loan on a six year old vehicle.
- The applicant selects the duration of the loan on one form (Collect Loan Info), and specifies the model year of the vehicle on a different form (Collect Vehicle Info).
- Our process presents the Collect Loan Info form prior to the Collect Vehicle Info form.

When the applicant requests a five-year used vehicle loan on an older car, they receive an error, as in the following example. Since PRPC evaluates the constraint declaratively, the applicant receives an error even though they have yet to enter the model year.

Since the model year is specified on a different form, the applicant can't resolve the violation easily. They would need to change the loan purpose just to resolve the violation. This would allow the applicant to advance to the Collect Vehicle Information form and enter an valid model year, and then return to the Collect Loan Information form to change the loan to a used vehicle loan.

If the root cause of the violation was a property that had been set declaratively, or our application did not use a screen flow, the applicant might be unable to resolve the violation *at all*.

Creating a Constraints Rule

Use the Application Explorer to access the constraints rules under the Decision category. Use the Rules Explorer to list all the constraints rules available to you. To create a constraints rule, in Application Explorer, right-click **Decision > New > Constraints**.

When Can You Use Constraints?

The properties to be constrained may be in the Applies To class or in the class of an embedded page.

To build for change, we may want to delegate a constraint rule to non-developers who then can make changes directly in the Constraints tab when business changes require an update to the rule.

Alternatives to Constraints

PRPC also provides the Validate rule to allow us to validate input data. **Unlike constraints, which are triggered automatically when a property value changes, a validate rule must be explicitly called**.

Validate rules test property values when a browser form is submitted and prior to running a flow action. Validate rules help end users enter valid property values into an HTML form or validate data received from another system or source. The application's user interface determines whether, when, and how the end user learns about this validation failure and how they are prompted to correct the input.

Module 12: Using Procedural Logic

Lessons Covered:

- Data Transforms
- Advanced Features of Data Transforms
- When Data Transforms Aren't Enough

Data Transforms

Objectives

At the end of this lesson, you will be able to:

- Describe data transforms
- Identify the key parts of a data transform rule
- Use a data transform to set initial property values

Things to Know

What is a data transform?

Generally speaking, data transformation involves both mapping data from a source to a target, and performing any conversions or other transformations on that data that might be required to achieve the intended mapped results.

A **data transform** defines how to transform source data values — data that is in one format — into data of another format (the 'destination', or 'target'). In a PRPC application, data transforms provide us with a way to easily copy, map, and transform clipboard data (pages and properties on those pages).

A data transform rule is a structured sequence of actions. When the system invokes the data transform rule, it invokes each action in turn, following the sequence defined in the data transform's rule form. Data transform rules are in the *Data Model* category in the Application Explorer.

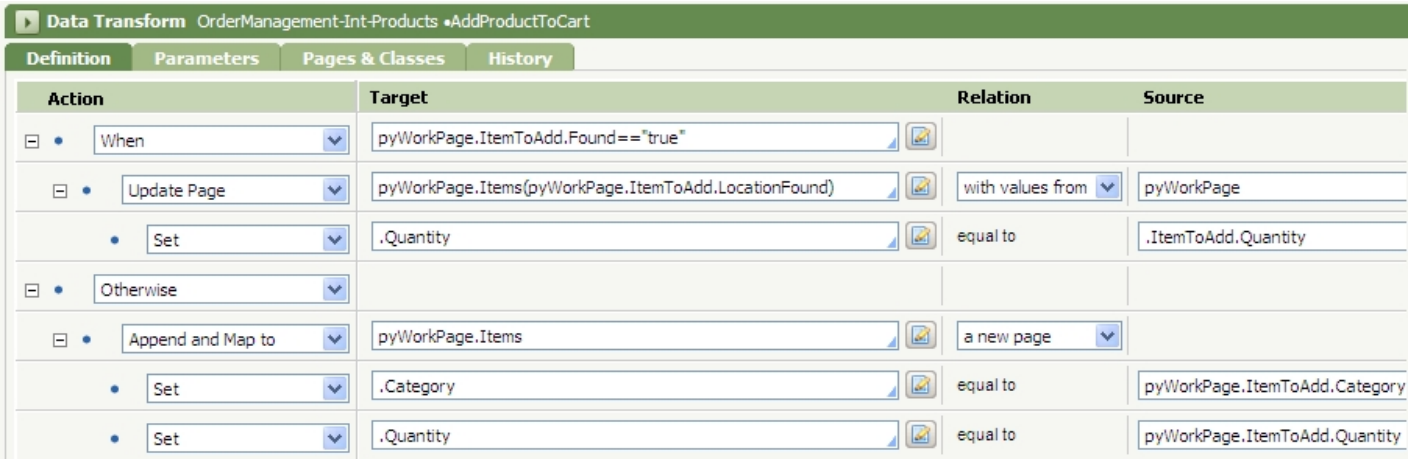
When should you use a data transform?

Below are some situations in which to use a data transform are when:

- Setting initial properties and their values on a given clipboard page
- Copying a clipboard page to make a new page
- Mapping properties (and their values) on one clipboard page to another, existing page
- Mapping properties (and their values) on one clipboard page to a new page
- Appending pages from one Page List property to another, or from a Page Group property to a Page List property
- Iterating over pages in a Page List property and mapping data on each page

Key Parts of a Data Transform Rule

The most important tab is the Definition tab. The sequence of actions to be taken is designed as rows in a tree grid. Except for the conditional types of actions (such as When) and transitional actions (such as Exit For Each), the system invokes each row in sequence, starting with the first row.



Each row provides for:

Column	Description
Action	An action to be taken.
Target	A target, if the action requires a target.
Relation	A relation, if the action requires one, between the target and source for that row.
Source	A source, if the action requires a source.

Some actions do not involve sources or targets or relationships, such as the Exit Data Transform action. In the form, you cannot select an item if it cannot be used in the specific context of the particular action in that row, or in the row's position in the grid with respect to its parent rows.

Best Practices

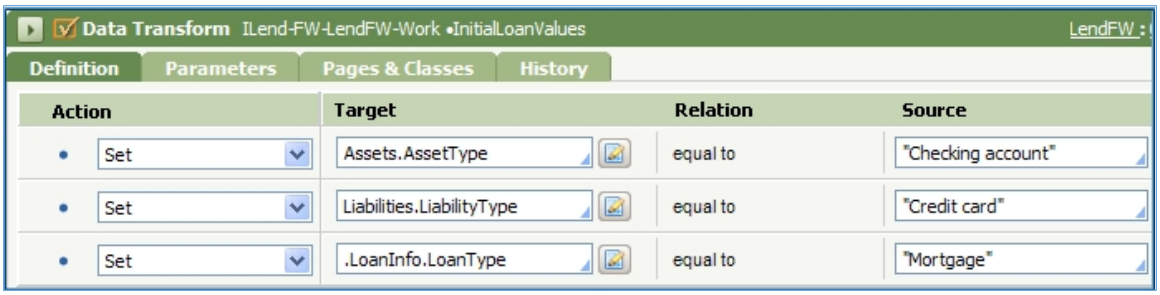
Aim to limit a data transform to fewer than 25 steps. If more actions are needed, create a separate data transform rule and use the Apply Data Transform action in the first data transform rule to invoke the subsequent actions.

Using a Data Transform to Set Initial Property Values

One of the most common uses of a data transform is to set initial or default values for properties. The Property rule form that defines a particular property does not include an initial default value for that property, because the most appropriate initial value can depend on the situation or application. We use a Set action in a data transform to set the property's initial value.

This data transform sets the value of three properties, the:

- **AssetType property** on the Assets page equal to the phrase "Checking account"
- **LiabilityType property** on the Liabilities page equal to the phrase "Credit card"
- **LoanType property** on the LoanInfo page equal to the phrase "Mortgage"

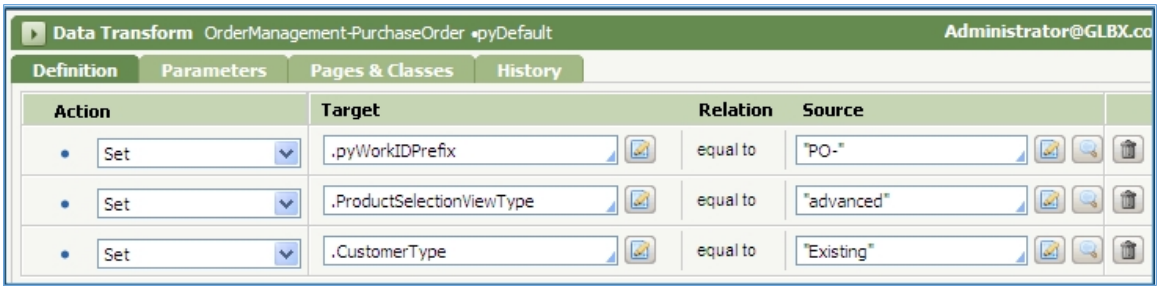


pyDefault Data Transform Rule

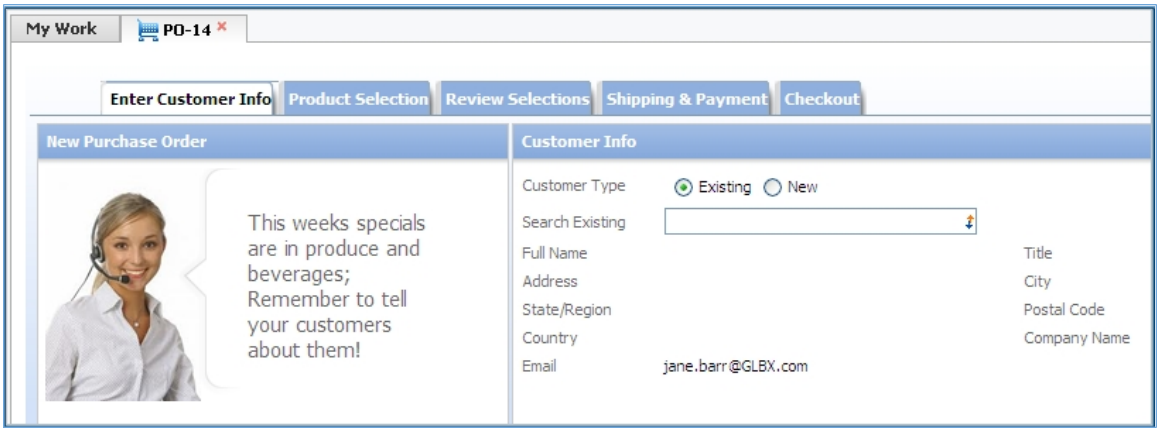
By default, various system wizards create a data transform rule named **pyDefault**. For example, when you use the Data Table wizard, it creates a pyDefault data transform rule for the data table's associated class. When the Application Accelerator generates the rules for the application, pyDefault data transform rules are created for each work type. By default, the work type's pyDefault data transform sets the work item's ID prefix to the value set in the Application Accelerator.

To initialize properties when a work item is initialized, update the pyDefault data transform rule in the work type's class to set the values for those properties. In the following example, the following initial values are set:

- The**pyWorkIDPrefix property** is set to "PO-", to set the work item prefix.
- The**ProductSelectionViewType property** is set to "advanced", so that in the work item form, the system initially displays the advanced Product Selection view to the user.
- The**CustomerType property** is set to "Existing", so that the radio button for the Existing choice is initially selected when the system displays the work item form.



When a customer service representative begins a new purchase order, the work item ID (PO-14) uses the PO prefix, and the **Existing** radio button is selected in the form.



Advanced Features of Data Transforms

Objectives

At the end of this lesson, you should be able to:

- Describe advanced features of data transforms
- Identify locations in an application where data transforms are commonly used
- Describe how to use parameters in a data transform
- Use a data transform to copy data from one page to another

Things to Know

Chaining Data Transforms

A data transform has a powerful option to **automatically apply any data transform in its ancestor classes that share the same name**. When the Call superclass data transform? checkbox is selected, PRPC uses pattern and directed inheritance to find and apply the data transforms in the ancestors. Then PRPC applies each of those same-named data transforms, starting with the one in the highest ancestor. The current data transform is applied last.

For example, if the pyDefault data transform in the PurchaseOrder-Work class has the checkbox selected, then at runtime the system locates the PurchaseOrder-Work class's immediate parent class (for example, Work-Cover), and then the next parent class (Work-), and then the next one (@baseclass), and identifies all of the data transforms named pyDefault in those ancestors. When the system reaches the highest ancestor and identifies the data transforms with the same name as the initial one, it applies the pyDefault data transform in the highest ancestor class (@baseclass), and then the next one (Work-Cover-), and then the next. The pyDefault data transform in the PurchaseOrder-Work class is applied last.

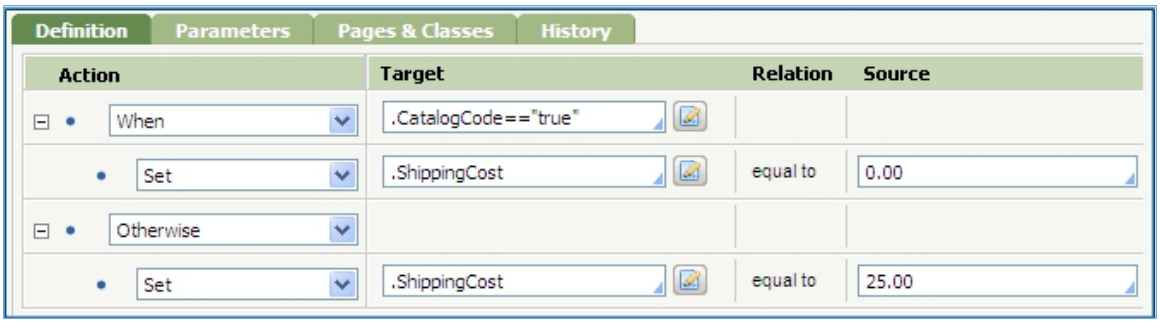
This **chaining** operation is especially helpful for setting initial values for starting processes for related work types, because properties that are common across the work types can be initialized in the pyDefault data transform at one layer, and properties that are distinct for each work type can be initialized in the pyDefault data transform in each work type's class.

Conditionally Setting Initial Values

Sometimes you want to initialize properties to particular values under a certain condition, and use alternative values in other conditions. For example, if a customer enters a catalog code, the cost of shipping is free. Otherwise, the normal shipping costs are applied.

To conditionally set values, use the data transform's **When and Otherwise actions**. In the When row, specify the condition that must be met. In the children rows of the When row, specify the actions that are taken when that condition is met. In the Otherwise row, specify the alternative actions.

In the following example, the CatalogCode property is a single value TrueFalse property. The user selects a checkbox in the work item if the user has a catalog code. The Where action in the SetShippingCosts data transform looks for whether the property is set to true ("true"), and if so, the ShippingCost property is set to value 0.00. If the CatalogCode property is not set (false), the Otherwise branch is used to set the ShippingCost to 25.00.



Actions Used in a Data Transform

The actions available in a data transform and their primary uses are:

- **Append to** — To copy a page from the source to a target page list.
- **Append and Map to** — To add a new page to a target page list, for subsequent actions to set or map properties on that new page.
- **Apply Data Transform** — To apply the actions defined in another (different) data transform to a clipboard page.
- **Comment** — To provide explanatory comments within the action sequence.
- **Exit Data Transform** — To exit the data transform at that point in the sequence. Typically used in a conditional branch to skip remaining actions based on a condition.
- **Exit For Each** — To stop the iterations of a preceding For Each Page In action.
- **For Each Page In** — To iteratively apply actions specified in its child rows to all of the pages in the specified target (page list or page group). In the child rows, use the <CURRENT> keyword as the index value to refer to the current page for the current iteration. For example, if the target of a For Each Page In is a page list named Items, in the child rows, you can refer to the current page of an iteration using .Items(<CURRENT>).
- **Otherwise** — To provide actions for the alternative to the When or Otherwise When actions.
- **Otherwise When** — To conditionally provide alternative actions to a previous When action.
- **Remove** — To delete the target and any associated values from the clipboard. The target can be a value mode property (Single Value, Value List, or Value Group), a page mode property (Page, Page List, or Page Group), a top-level page, or a declarative page.
- **Set** — To set the target equal to the source (which can be a top-level page or a property). You can set Single Value properties, top-level pages, embedded pages, page lists and page groups. The Set action is typically used to initialize properties. For example, you can use the Set action to copy a value (specified in the Source field) to another property (specified in the Target field). That value can be a property or a literal string (such as "WO-"). The Set action can also be used to copy a page property from another page property, or a top-level page, as well as to copy a page list or page group property to another page list or page group property.
- **Update Page** — To set the target context, source context, or both for the subsequent child action rows.
- **When** — To conditionally perform actions.

For data transforms that have many parent-children combinations of actions, you can use the right-click context menu options to more easily work with the action array as you design the data transform. You can also drag and drop rows (using the blue dots at the left of the rows) to reorder them within the array.

For more information on using the action rows in the data transform rule form, see the Pega Developer Network (PDN) article PRKB-26379 *Introduction to data transforms*.

Locations Where Data Transforms Are Commonly Used

Various rule types can specify data transform rules, so that the appropriate data values are used at the appropriate time in the work process.

- **Starting processes** — Specify a data transform rule on the flow rule's Processes tab to set initial properties for the work item when the process starts.
- **Flow actions** — Specify a data transform on the flow action's Action tab, to perform data transform actions before or after displaying the flow action's user interface to the user.
- **Process Modeler Connector shapes** — Specify a data transform to perform data transform actions before the subsequent step in the process.
- **User interface elements that use the following items**:
    - Refresh This Section — To refresh the section with values set in the data transform.
    - Refresh When — To refresh with values set in the data transform.
    - Refresh Section, Show Harness, and Open URL In Window on-click actions.
- **Other data transform rules** — Specify using the Apply Data Transform action.
- **Activities** — Specify in methods such as the Apply-DataTransform method, Page-New, and Page-Copy.

Using Parameters in a Data Transform

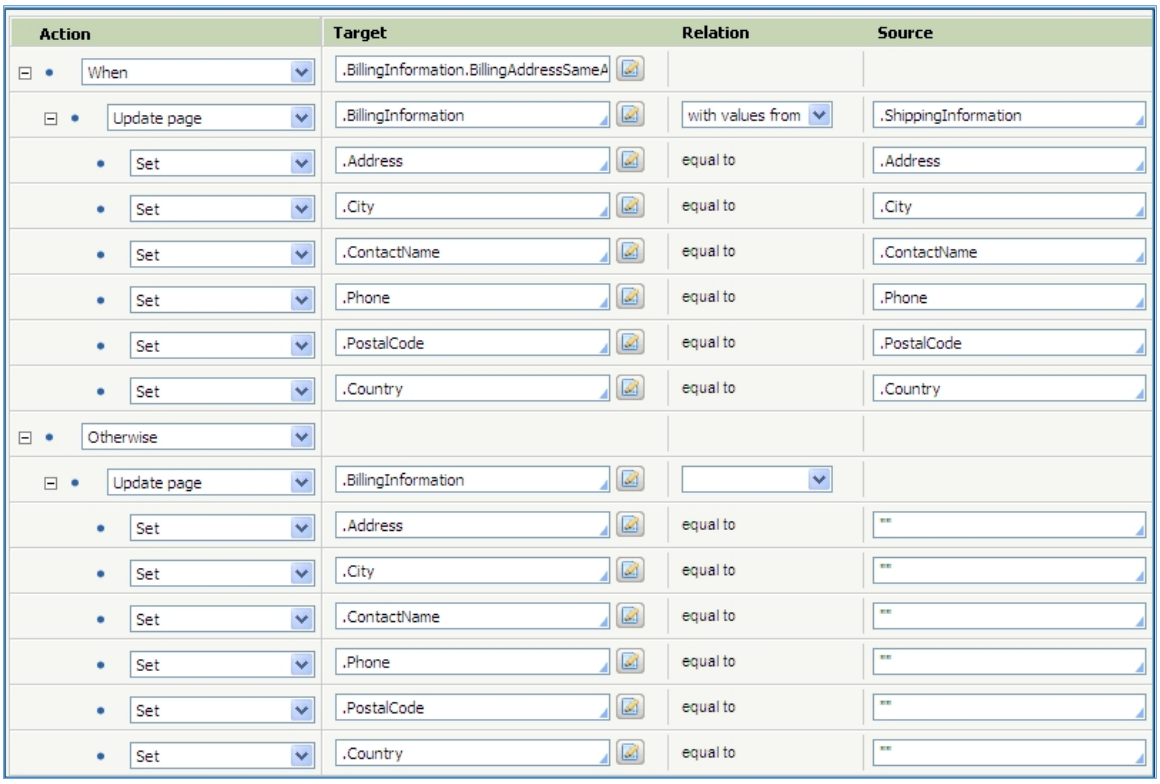
Parameters are variables used as inputs or outputs for the data transform. They are defined on the Parameters tab of the data transform's rule form, and can be used in an action row by using the keyword param, followed by the name of the parameter. For example param.Address.

You can pass parameters into a data transform when specifying that data transform in one of the following places:

- Apply Data Transform action in a data transform rule.
- Activity methods that use a data transform (such Apply-DataTransform or Page-New).

Using a Data Transform to Copy Data From One Page to Another

A common and powerful use of a data transform is to copy data values from a clipboard page to another page. This example involves two page properties: BillingInformation and ShippingInformation. On the work item form is a checkbox to optionally specify that the customer's billing address is the same as the shipping address. If the checkbox is selected (BillingAddressSameAsShipping is true), then the values on the BillingInformaton page are initialized to values from the ShippingInformation page. Otherwise, when the checkbox is unselected, the properties on the BillingInformation page are initialized to blanks.



Action	Target	Relation	Source
When	.BillingInformation.BillingAddressSameA		
Update page	.BillingInformation	with values from	.ShippingInformation
Set	.Address	equal to	.Address
Set	.City	equal to	.City
Set	.ContactName	equal to	.ContactName
Set	.Phone	equal to	.Phone
Set	.PostalCode	equal to	.PostalCode
Set	.Country	equal to	.Country
Otherwise			
Update page	.BillingInformation		
Set	.Address	equal to	""
Set	.City	equal to	""
Set	.ContactName	equal to	""
Set	.Phone	equal to	""
Set	.PostalCode	equal to	""
Set	.Country	equal to	""

When Data Transforms Aren't Enough

Objectives

At the end of this lesson, you will be able to:

- Describe activities
- Identify alternatives to creating activities and the reasons to use the alternatives instead of creating activities
- Describe the key parts of an activity

Things to Know

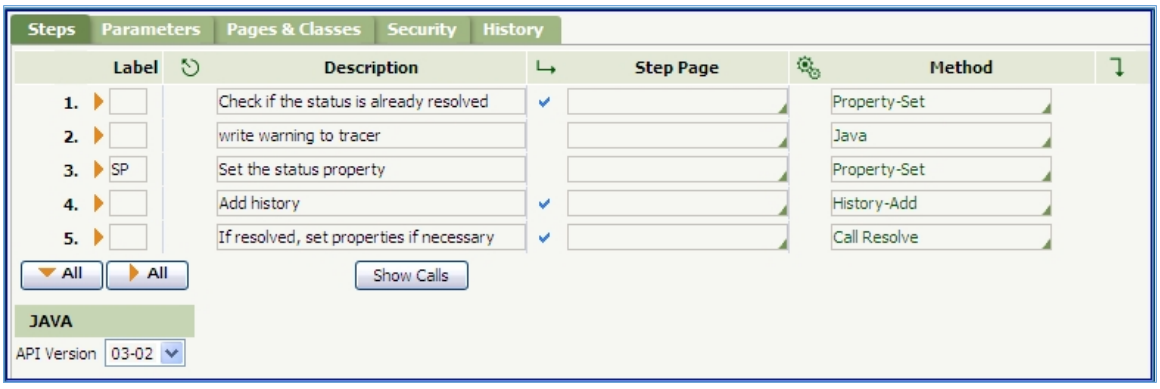
What is An Activity?

An activity is a rule that corresponds in many ways to a traditional, procedural program written in a programming language. Like a data transform rule, an activity rule consists of a sequence of steps that are typically performed in order, from the top one listed on the activity rule form to the bottom one. Each step of an activity contains an atomic operation, known as a **method**. For example, the Obj-Open method retrieves a single instance stored in a database and converts the retrieved instance to a clipboard page. Typical uses of activities within an overall process include:

- Instantiating work items
- Routing a work item to a worklist or workbasket
- Creating and processing e-mail (correspondence)
- Invoking connector rules to query database tables and update them, or send and receive information to and from web service endpoints

Like a data transform rule, an activity rule:

- Is typically used for data manipulation
- Is structured as a series of steps, executed in sequence
- Can be called from a flow action as a pre- or post-processing step
- Has a default data page called its **primary page**, which is a clipboard page of the same class as the activity rule
- Can use parameters as inputs and outputs
- Provides for iterations, conditional operations, and transitions (such as jumps between steps)



Activity rules are found in the *Technical* category in the Application Explorer.

When Should You Use An Activity?

Activities should be used rarely. In general, it is best to use other rule types instead of activities. Because an activity can call custom Java code, it is harder to maintain than the alternatives. Also, because an activity can perform complex actions and sequences, it can be difficult to read and debug an activity originally created by someone else. Other rules types can provide more focus and structure, and are easier to maintain, such as:

- Report Definition rules
- Data transforms
- Declare expressions
- Declarative pages
- Decision tables, decision trees, and map value rules

For example, instead of using a series of activity steps for automatic calculations when an input property changes, use a Declare Expression rule to calculate declaratively.

If, after seeking to use another rule type, we find that a specific situation cannot use a rule type other than an activity rule, we can locate and use one of the API activities that are available in PRPC. Use the APIs gadget on the Processes landing page to see these standard API activities. Don't create an activity in your application when you can call a standard activity to perform the same task.

Examples of standard activities are:

- **AddWork** — Creates a new work item
- **CorrNew** — Creates a new email (correspondence)
- **SetOwner** — Sets the pyOwner of a work item
- **Update Status** — Updates the status of a work item
- **UpdateWorkObject** — Update a work item
- **pxTransferCaseOwnership** — Transfers a case and its children to a new owner

Who Should Create Activities?

Upon determining that the situation has such unique context that another rule type cannot be used and that an API activity does not suffice, a Lead System Architect (LSA) or Senior System Architect (SSA) might create an activity rule.

In this rare case, follow these best practices:

- Limit the activity to fewer than 25 steps.
- Avoid custom Java. Use PRPC's standard methods, functions, and activities (using the Call activity method) instead.
- Avoid hard-coding values in the activity rule form. Use parameters to pass names such as class names and flow names. Use parent classes such as Work-Cover-, rather than specific classes on the Pages & Classes tab.
- Where possible, save the activity to a generalized RuleSet. For example, activities which can be used in multiple applications should be saved to the framework layer for those applications or to a common RuleSet so they can be reused as needed.

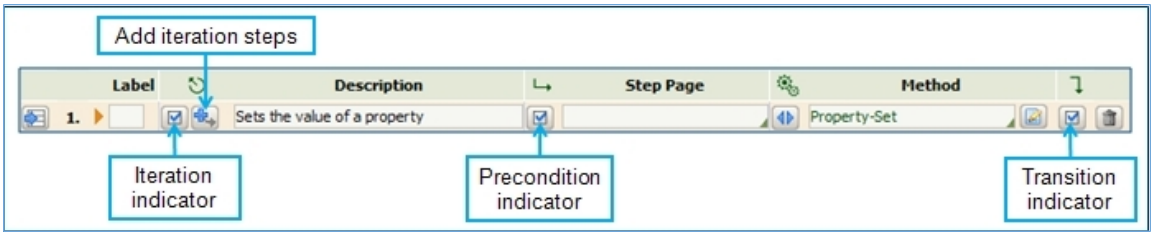
Key Parts of an Activity

Except as controlled by preconditions and transitions, the system processes the steps of an activity in sequence, from the first step at the top of the rule form to the last step at the bottom.

Steps

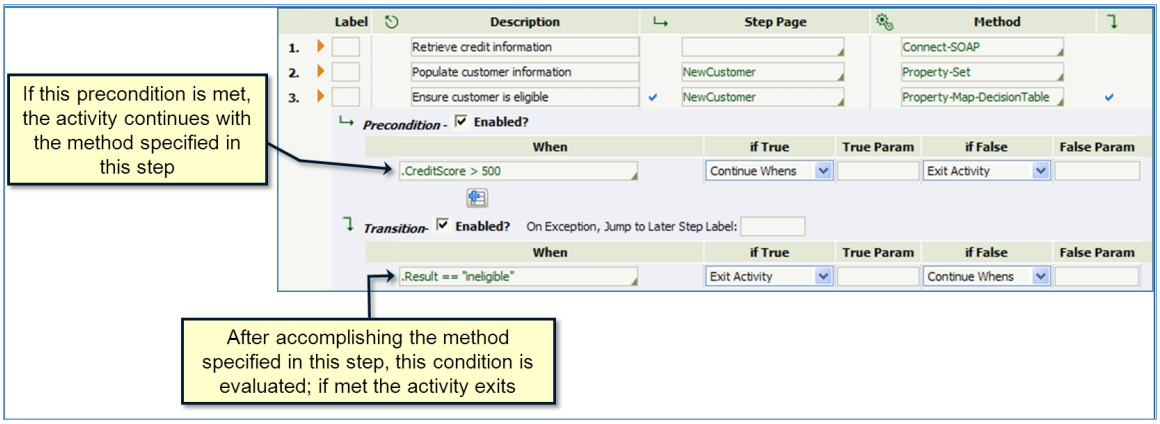
Each activity step can execute:

- A standard method
- Custom Java code
- Another activity



Optionally, each step can include:

- **Iterations** — Use to execute the step multiple times, or until a condition is met. (Similar to the 'do while' or 'repeat until' structures of programming languages.)
- **Preconditions** and **transitions** — Use to skip steps or perform additional processing based on certain conditions. Preconditions execute before the step is processed. Transitions execute after the step is processed. For both preconditions and transitions, the system tests a condition that evaluates to True or False, and directs subsequent processing based on the result.



Parameters and Local Variables

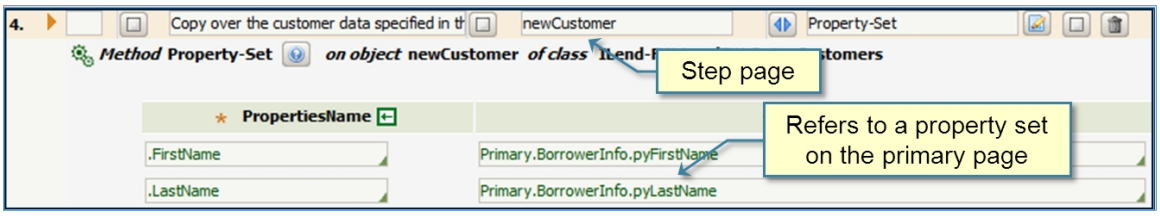
Parameters are variables used as inputs or outputs for the activity. They are defined on the **Parameters** tab of the activity's rule form, and can be passed to a method or activity either individually or by page.

Local variables are scalar-only variables stored in the Java generated for the activity, rather than on a clipboard page.

Primary and Step Pages

The activity's primary page (default data page) is a clipboard page with the same class as the activity. The system automatically generates this page when calling the activity, and executes the activity's sequence of steps within the scope of the primary page. Use the keyword Primary to refer to the activity's clipboard page.

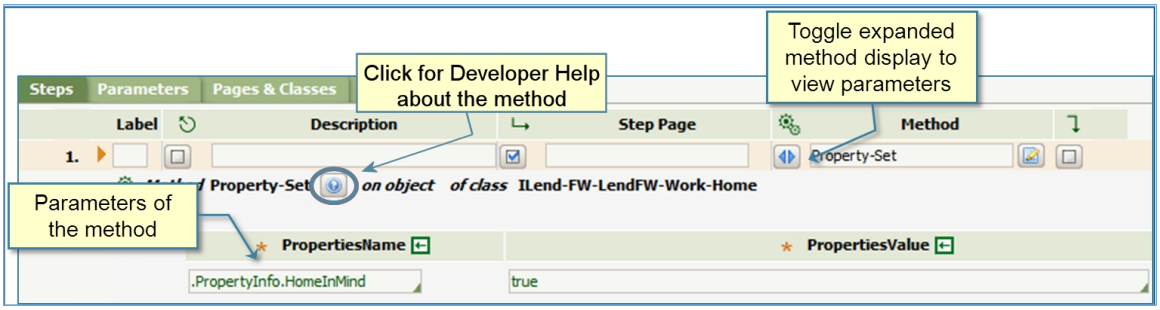
A **step page** is a clipboard page referenced by a specific step. When a step references a step page, the system executes that step's method within the scope of that step page, and uses that page to store data for that specific step action. If a step does not specify a step page, the system executes the step's method on the activity's primary page.



Methods You Will See Most Often

When you open the rule form for an activity, the methods commonly seen in the activity steps are:

- **Page-New** — Creates a new page in the clipboard. (A data transform can create a new page also.)
- **Page-Remove** — Removes a page from the clipboard, to save memory when the page is no longer needed. (A data transform can remove a page also.)
- **Property-Set** — Sets the value of a property. (A data transform can set a property value also.)
- **Obj-Open** — Opens an object from a database and loads it onto the clipboard.
- **Show-Page** — Displays the content of the page in XML (useful for debugging an activity).
- **Call Activity** — Executes the specified activity (useful for reusing activities).



Module 13: Integrating with External Data Sources

Lessons Covered:

- Exchanging Data with Other Applications
- Requesting Data from an External Source
- Responding to External Requests

Exchanging Data with Other Applications

Objectives

At the end of this lesson, you will be able to:

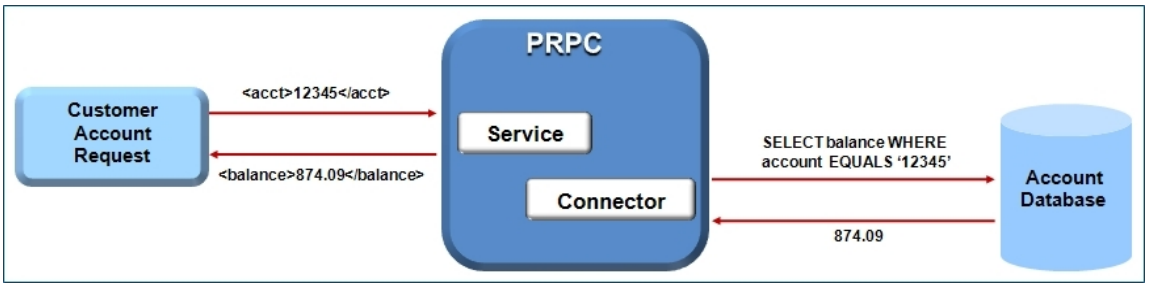
- Describe PRPC options for exchanging data with other applications
- Explain the difference between a connector and a service

Things to Know

PRPC's integration facilities provide multiple ways to control communication and messaging between our application and other systems and databases. External systems can query our application for data, or our application can query an external system. Data may be mapped directly to properties or parsed and transformed. The application can serve as the endpoint for an external connection — as a means to provide data to another system.

PRPC's integration facilities provide a set of protocols contained within a toolkit that allows users to quickly create connections to — and endpoints for — external systems. By integrating with other systems, designers can leverage existing applications and services. External systems can also send information to Process Commander, and a work item can be created based on this information.

Let's first look at the two most-basic forms of PRPC integration: *connectors* and *services*.

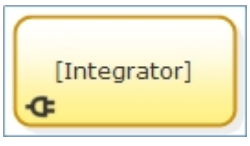


Connectors

Connectors represent integrations where PRPC initiates a request to an external system. PRPC uses a common protocol — such as SOAP or HTTP — to exchange data with the external system, and maps the data exchanged to application properties. This allows us to enhance the functionality of our applications in various ways, such as:

- Querying a state database to review driving records and establish an insurance premium.
- Determining if a house lies on a flood plain and is eligible for federal flood insurance.
- Verifying the existence and status of financial accounts.
- Looking up currency conversion rates to convert an account balance to Pounds, Euros, or Yen.

Connectors are generally called with an **Integrator** shape.



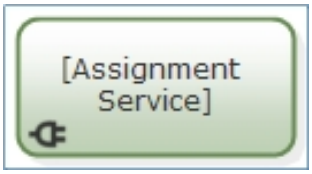
The Integrator calls an activity, which in turn calls a connector rule that defines the request. Once the information is returned to PRPC, the process advances according to the flow logic.

We can create connectors to interface with the following services:

- BPEL (**B**usiness **P**rocess **E**xecution **L**anguage)
- CMIS (**C**ontent **M**anagement **I**nteroperability **S**ervices)
- EJB (**E**nterprise **J**ava**B**ean)
- HTTP (**H**yper**T**ext **T**ransport **P**rotocol)
- Java
- JCA (**J**ava EE **C**onnector **A**rchitecture)
- JMS (**J**ava **M**essage **S**ervice)
- MQ (IBM WebSphere MQ messaging)
- .NET
- SOAP (**S**imple **O**bject **A**ccess **P**rotocol)

Services

A service operates as the reverse of a connector. **Rather than the query originating with PRPC, it originates with the external system. PRPC responds to the query with the relevant data.** This request either launches a process — which may create a work item — or enters a process at an **Assignment Service** shape.



If the service accesses an Assignment Service shape, the process pauses at the shape until PRPC receives the service request.

PRPC supports the following types of web service connections:

- BPEL
- COM (Microsoft **C**ommon **O**bject **M**odel)
- Email
- EJB
- File
- HTTP
- Java
- JMS
- JSR94 (**J**ava **S**pecification **R**equest 94)
- MQ
- Portlet
- SOAP

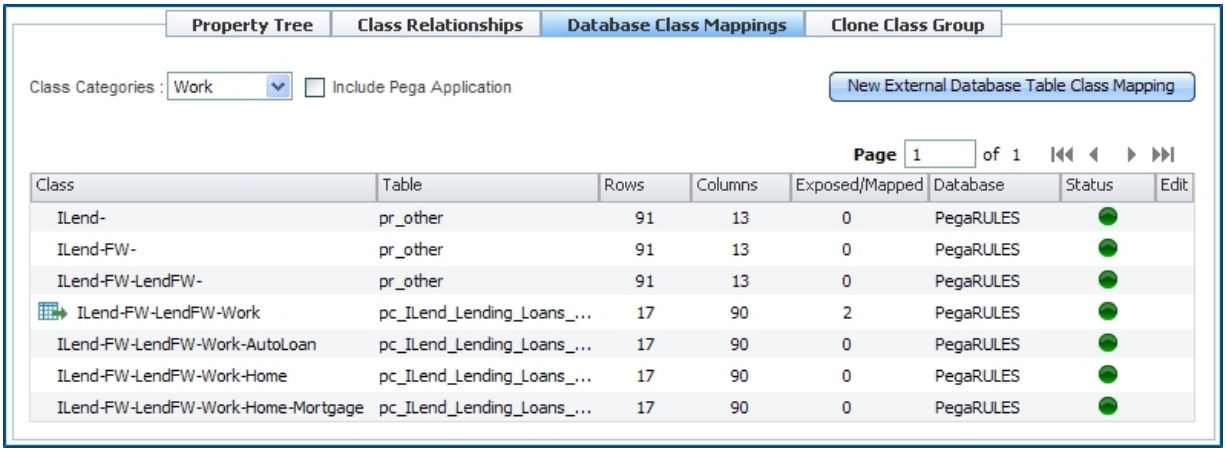
Other Integration Options

Now that we've discussed connectors and services, let's look at some of the other integration options available in PRPC.

External Database Table Class Mappings

When connecting to an external SQL database, we can avoid the need for a connector by using the **Database Class Mappings** gadget. If we know specific information about the data store on the other end of the connection — the name of the database and the appropriate table — we can query it directly, without needing to configure a connector. Effectively,

the class mapping acts as a connector using the SQL protocol. This gadget is available from the Classes and Properties landing page, and lists all of the mappings between applications and database tables.



Class	Table	Rows	Columns	Exposed/Mapped	Database	Status	Edit
ILend-	pr_other	91	13	0	PegaRULES		
ILend-FW-	pr_other	91	13	0	PegaRULES		
ILend-FW-LendFW-	pr_other	91	13	0	PegaRULES		
ILend-FW-LendFW-Work	pc_ILend_Lending_Loans_...	17	90	2	PegaRULES		
ILend-FW-LendFW-Work-AutoLoan	pc_ILend_Lending_Loans_...	17	90	0	PegaRULES		
ILend-FW-LendFW-Work-Home	pc_ILend_Lending_Loans_...	17	90	0	PegaRULES		
ILend-FW-LendFW-Work-Home-Mortgage	pc_ILend_Lending_Loans_...	17	90	0	PegaRULES		

Listeners

Listeners are used in conjunction with services, and allow us to react to changes to a queue as inputs enter. Listeners operate in the background, monitoring a location — such as a file system, email, or JMS queue — for requests. Once a change in the monitored location is detected, the listener triggers a corresponding service rule to carry out the actual integration functionality.

Requesting Data from an External Source

Objectives

At the end of this lesson, you will be able to:

- Connect an application to an external data source
- Request data from an external source

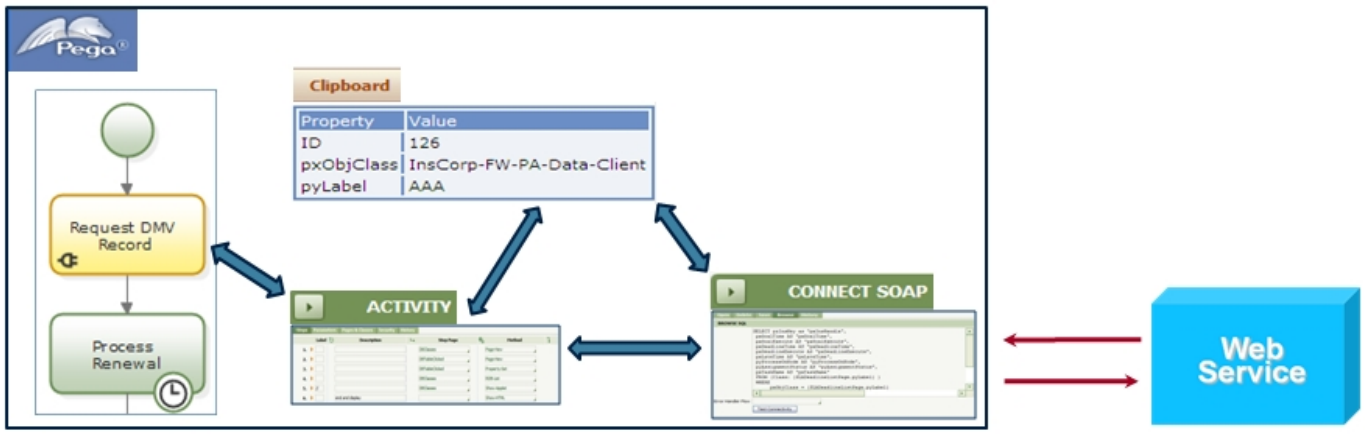
Things to Know

Previously, we discussed the basics of integration in PRPC. In this lesson, we'll discuss in greater depth how to use data from an external source in our application.

Connectors

Connectors allow us to obtain data from an external data source. They are used frequently in flows, using the Integrator shape, to request data at a particular point. The Integrator calls an activity, which uses the information provided in a connect rule to open a connection to the external data source and request data. The data store returns the requested data, which is written to the clipboard — from which it can be used by the application — the activity finishes processing, and the process advances to the next step.

The following example illustrates how a process requests data from a web service, using a SOAP (**S**imple **O**bject **A**ccess **P**rotocol) connection.



So, to access data from an external data source — in this case, a web service — in a flow, at a minimum we need:

1. An Integrator shape in the flow.
2. An activity to be called by the Integrator.
3. A Connect SOAP rule to be called by the activity.

While we could create the activity and connect rules (connect SOAP, in this example) manually, the easier way is to use the **Connector and Metadata Accelerator**, available from the Integration Overview landing page.

Using the Connector and Metadata Accelerator

We use the Connector and Metadata Accelerator to import information about an external application or system. The accelerator processes this information to generate the rules and data objects that PRPC applications can use to communicate with that external system, either **connector rules** or **data mapping rules**.

Generating Connector Rules

Select this option to generate the rules and data objects necessary for an application to send messages to or make requests of external systems and then process the results or response. The accelerator can generate rules for **only** the following connector types, which communicate with external systems through protocols or technologies that can expose information or metadata about the construction of the application:

- EJB (**E**nterprise **J**ava **B**eans)
- Java
- .NET
- SOAP

Depending on the connector type, the accelerator either parses a WSDL (**W**eb **S**ervice **D**efinition **L**anguage) file or imports Java class metadata through introspection. It then converts the resulting information into the following to build the connector:

- Class and property rules to support the objects or data described in the WSDL, Java class, or EJB
- One connector rule for each method or operation selected
- One connector activity for each connector

For SQL (database) connections, we can avoid using a connector altogether by creating an external database class mapping. For all other protocols, we must create the rules and data objects manually.

Generating Data Mapping Rules

Select this option when you want to either:

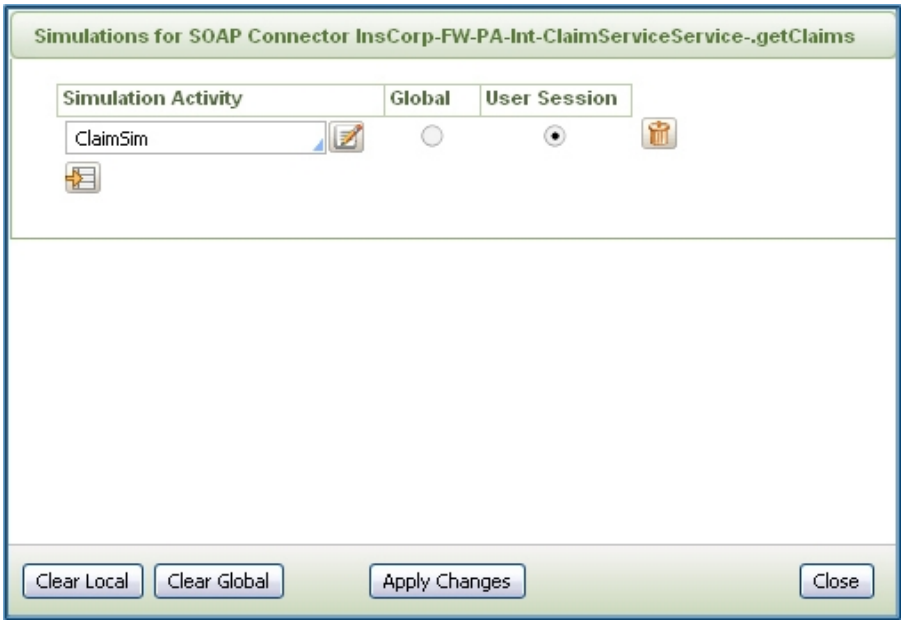
- Generate Parse XML, XML stream, class, data transform, property, and other rules that implement in PRPC the data model of an external system based on the schema described in an XSD or DTD file.
- Generate class, data transform, and Java property rules that reflect the data model described by an EJB. In this case, the accelerator also generates Java wrapper classes that access the get and set methods from the source Java class.

Testing Connectors

Connect Simulator

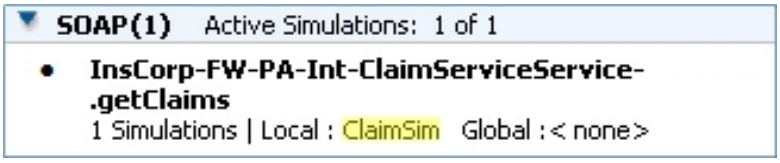
Once we've built our connector, we can test it without needing to connect to the actual data source. This is especially helpful when testing an application on an isolated system, or using data sources that charge for responses. To test a connector, we can use the **Connect Simulator**.

To use the Connect Simulator, we need to create an activity that provides property values that we would normally receive from the external source, through the connector. On the connector rule form, click **Simulations** on the **Service** tab to specify the activity, and select either the **Global** or **User Session** options. Once we do this, the connector executes the activity rather than connect to the external source when called from the flow.



We can override the simulator activity and restore the connector — either temporarily or permanently — with the **Connection Simulation** tool found in the Integration category of the Pega menu.

Active simulations are indicated on the Integration Overview landing page.



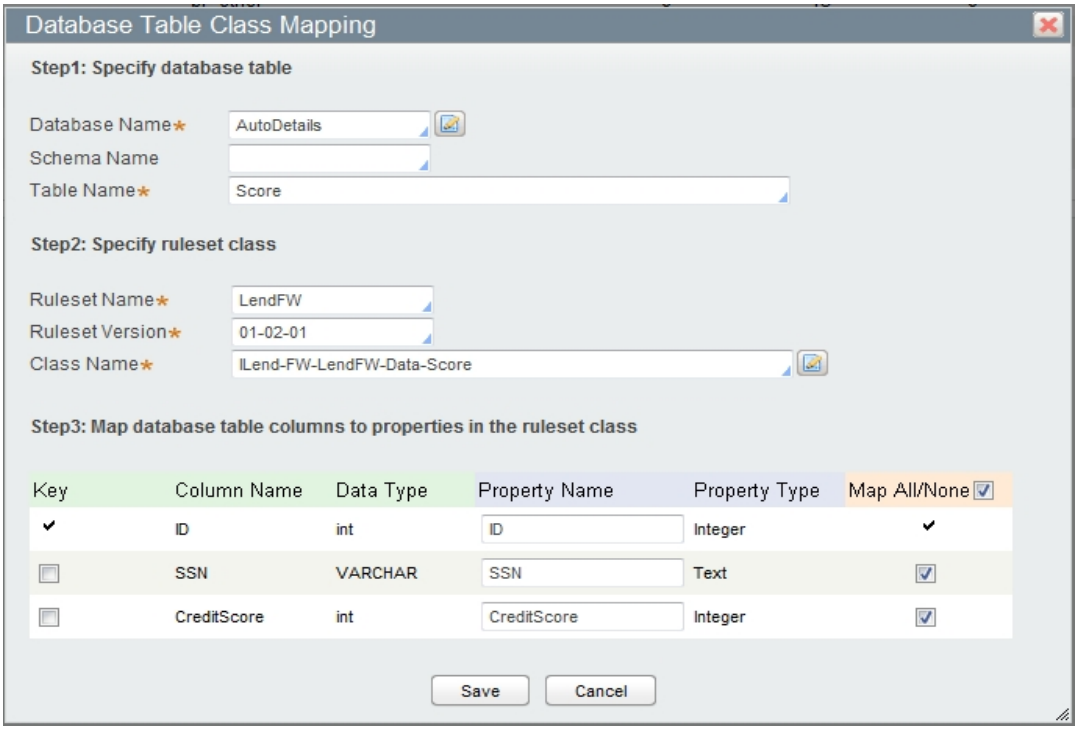
External Database Class Mappings

The External Database Class Mappings tool allows us to create a mapping between a class and an external database table. This allows direct access to the database table.

Using the External Database Table Class Mapping Tool

We use the External Database Table Class Mapping tool to map a class within our application to the external database table, in three steps:

1. Specify the database name and table.
2. Specify the class and RuleSet for the data in PRPC.
3. Map each desired database column to a property in the class and RuleSet provided in step 2.



The information provided in steps 1 and 2 is saved on the External Mappings tab of the class rule. This eliminates the need for a connector completely — and with it, the Connector and Metadata Accelerator.

Responding to External Requests

Objectives

At the end of this lesson, you will be able to:

- Describe how services work

Things to Know

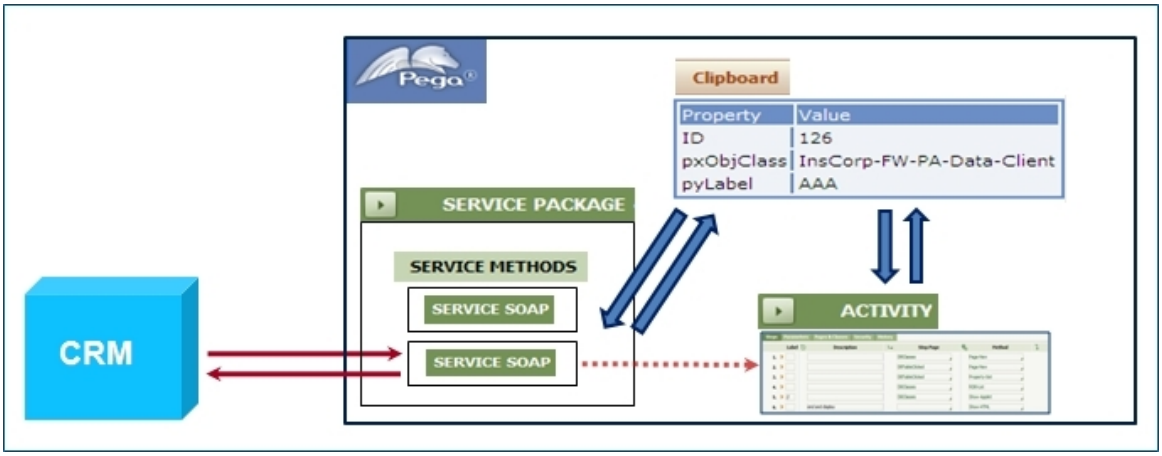
Now that we've discussed using external data in our application, let's explore the reverse: allowing other systems to access PRPC applications.

Services

As we discussed earlier, a service is the reverse of a connector; while connectors process *outbound* requests, **services process *inbound* requests**. In order to exchange data between two systems, both a connector and a service must be in place to create a connection and exchange data through a common protocol.

In the previous lesson, we learned how to create a connector in PRPC to communicate with a service. In this lesson, we'll discuss how to create a service to process inbound requests.

In PRPC service rules — sometimes referred to as *service methods* — respond to incoming requests by calling an activity, which processes the request and writes data to the clipboard. The requested data is then returned to the external agent that initiated the request. In the following example, an external CRM system queries a PRPC application for information about a customer, using a SOAP (**S**imple **O**bject **A**ccess **P**rotocol) service.



PRPC provides two standard service activities for services generated by the Service Accelerator.

- **svcAddWorkObject** — creates the work item and starts the flow. This is the standard service activity for services that create new work items.
- **svcPerformFlowAction** — identifies the work item and then calls the flow action. This is the standard service activity for services that perform a flow action on a work object.

A service does not have to create new work objects or perform a flow action; it can invoke any service activity that we choose. If the service activity does not already exist, the Accelerator generates a stub activity that we can then edit as necessary.

Each service is bundled into a service package. This package contains all of the rules — such as services, activities, and data transforms — that make up the service, and is used to generate the deployment file — such as a WSDL (**W**eb **S**ervice **D**efinition **L**anguage) file — that is distributed to facilitate connections from external sources.

Using the Service Accelerator

We can use the Service Accelerator to create the rules and data objects for the following service types:

- SOAP
- .NET
- HTTP (**H**yper**T**ext **T**ransport **P**rotocol)
- EJB (**E**nterprise **J**ava**B**ean)
- Java
- JSR94 (**J**ava **S**pecification **R**equest 94)
- JMS (**J**ava **M**essage **S**ervice)
- MQ (IBM WebSphere MQ messaging)
- File

For email services, we use the **Email Accelerator**. For portlet, BPEL, COM, and CORBA services, we must create the rules and data objects manually.

The Service Accelerator allows us to create services that do one of the following:

- Create and manage work
- Process input or output data
- Call an existing activity

The Service Accelerator creates all of the rules necessary to establish the service, which can then be accessed from an external application or system. For example, when creating a SOAP service, the accelerator creates **Parse XML** rules to translate between the incoming request (structured to match the WSDL) and our application's internal data structure (classes and properties) and **Stream XML** rules to structure the outgoing response to match the WSDL.

Reference Material

- PDN article PRKB-25074 — Using the Service Accelerator
- PDN article PRKB-25085 — About email
- PDN article PRKB-25109 — About JSR 168 Portlet Services

Module 14: Creating Your User Interface

Lessons Covered:

- UI Rules
- Advanced UI Controls

User Interface Rules

Objectives

At the end of this lesson, you should be able to:

- Describe the different components of the Harness
  - Harness Structure
  - Standard Harnesses
  - Perform Harness
  - Harnesses and Flows
  - Setting a Harness
- Describe how Flow Actions and the User Interface interact
- Review a User Interface rule
- Use the HTML Rules Inspector to identify User Interface Rules

Things to Know

UI Components

Three main rule types comprise the User Interface (UI) category:

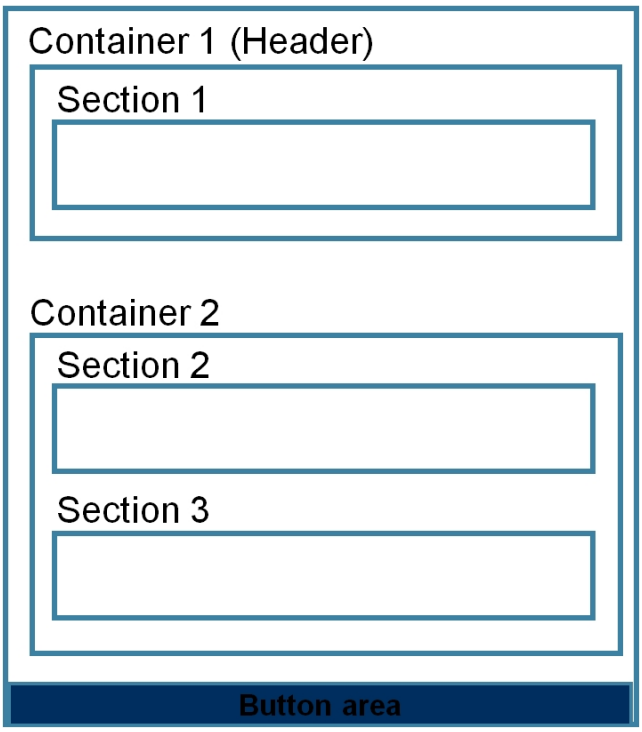
- **Harness**
  - Defines the top-level structure of a work item form
  - Calls section rules, which contain the UI components
- **Flow Action**
  - Presents the interface elements (forms to display, disposition options for work) to complete an assignment
  - Must reference either a section or HTML rule, or contain no user interface at all
- **Section**
  - Contains UI elements — properties, labels, controls
  - Are embedded in flow action or harness rules
  - Can be referenced by other UI rules

Harnesses

Harnesses provide a UI skeleton for the process.

Harness Structure

The hierarchy of components that make up a harness, from largest to smallest is containers (not covered in this course), which contain sections, which contain layouts, which contain cells. A section is the main visible component of a harness and contains icons, buttons, fields, labels, etc.



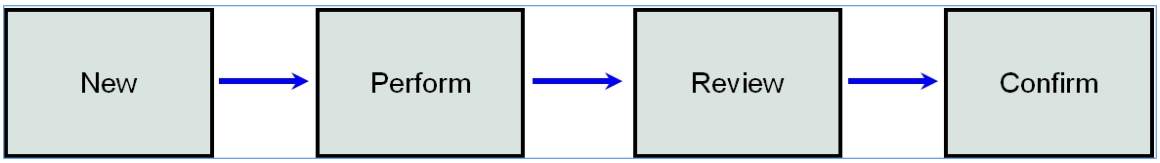
The body of the harness contains one or more containers or sections. You can add one or more sections to each container that you define in the harness. Format a container as a work item header, displaying standard icons in the title bar. Standard icons include Add attachments, View work item history,and Close the work item. We can use the button area to add standard buttons at the bottom of the harness. Wee can also create our own buttons to execute custom logic.

Standard Harnesses and Sections

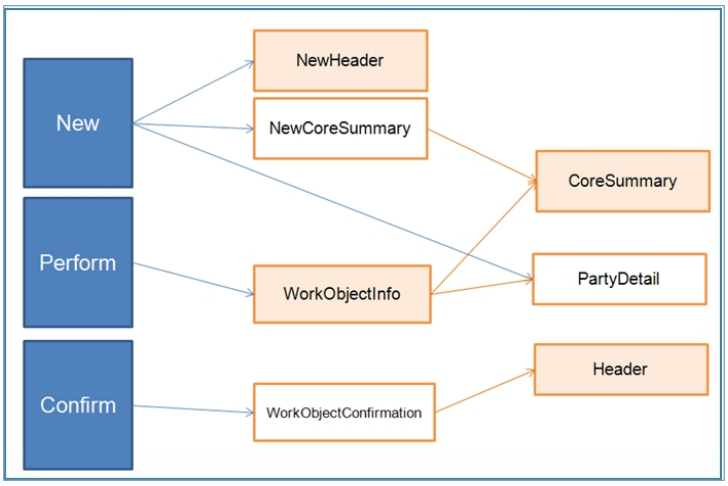
The Application Accelerator (AA) provides an application with several standard harness rules:

- **New** — Creates a new work item at the start of a process
- **Perform** — Allows users to select a flow action, completing an assignment
- **Review** — Presents the work item and assignment in read-only mode
- **Confirm** — Acknowledges user completion of an assignment with a read-only confirmation display of the work item

These standard harnesses appear in the following order in the process.

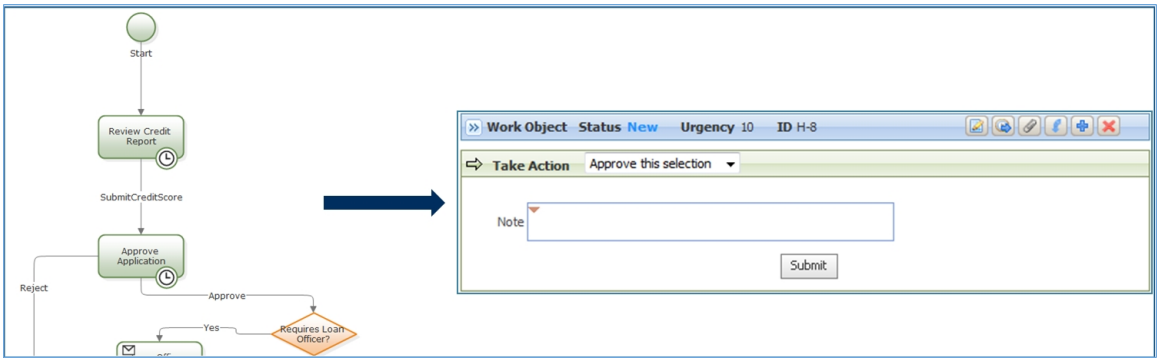


The harnesses provided by the Application Accelerator reference sections that are either shared among harnesses or referenced in other sections. In the following diagram, the shaded sections can be found in our application while the sections that are not shaded are inherited from Work-.



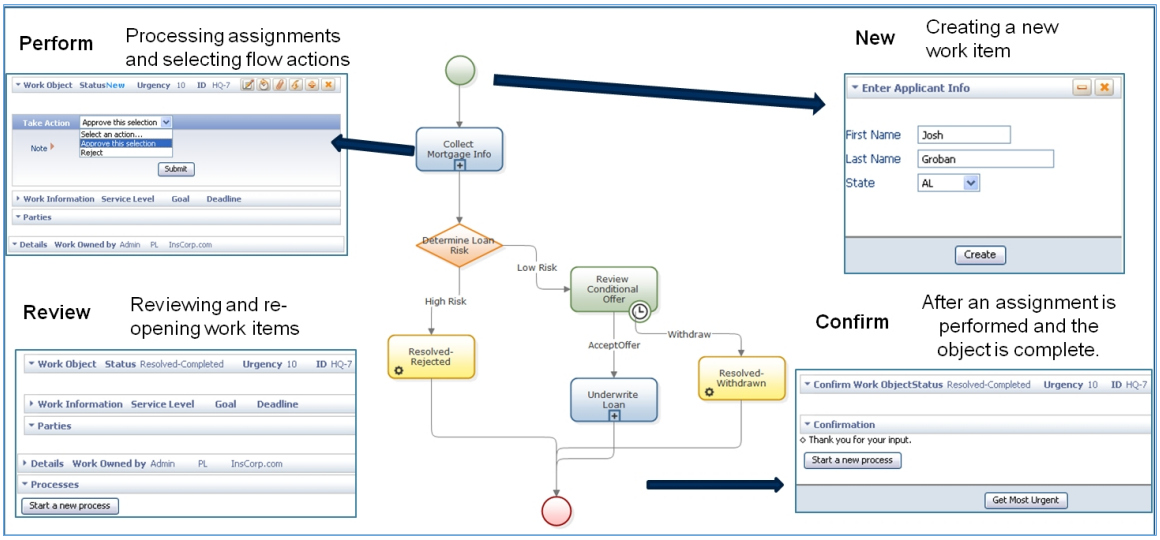
Perform Harness

Specified on an assignment, the Perform harness contains an action section, which presents the connector or local flow actions available on the assignment. When an action is selected, the action section displays the selected flow action.



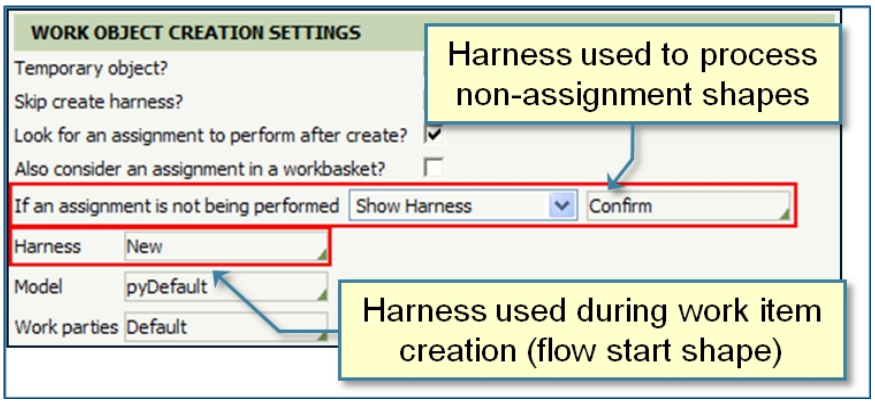
Harnesses and Flows

Different flow shapes can call different harnesses.

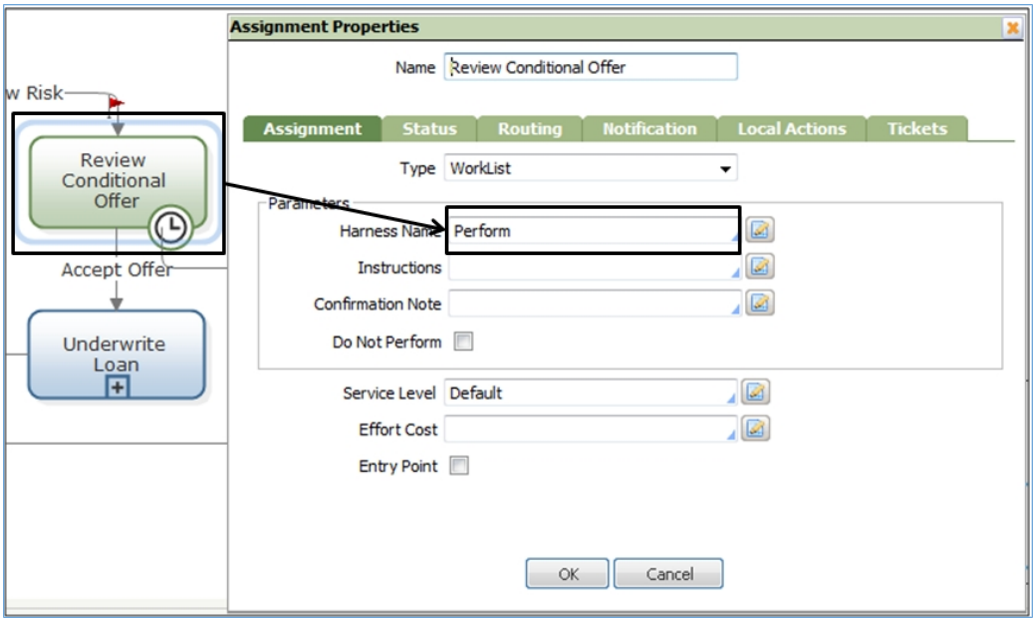


Setting a Harness

Set the default harness(es) for the starter flow on the Process tab of the flow rule.



Additionally, you can set a harness on an Assignment shape or the Start shape in a subflow.



Flow Actions

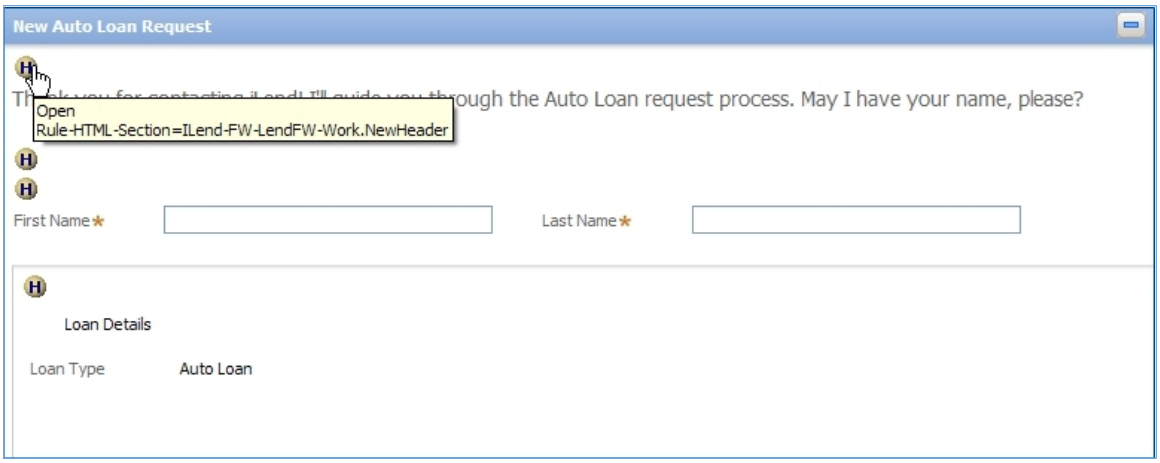
Flow Actions cannot directly contain UI features. Instead, they must reference a section or other HTML rules. This enforces the reusability of your UI. The Layout tab displays the UI rule that will be used at run time. Drag a section onto the Layout tab to associate it with the flow action.

Previewing a UI Rule

Click the Preview button  on the toolbar to view the UI component in isolation. This allows us to quickly review the layout and check the appearance of fields and other controls.

The HTML Rules Inspector

The HTML Rules Inspector provides an easy way to identify and locate rules present in the user interface at run-time. It identifies HTML rules such as harnesses and sections with an icon  . We can hover over the icon to determine the rule referenced, and we can click the icon to open the UI rule for editing.



Other Rules Inspectors allow us to identify property references in our UI, including those properties set with declare expression rules.

Advanced UI Controls

Objectives

At the end of this lesson, you will be able to:

- Understand when and how to use AutoComplete UI controls
- Understand when and how to use Dynamic Select UI controls
- Understand when and how to use Client Events

Things to Know

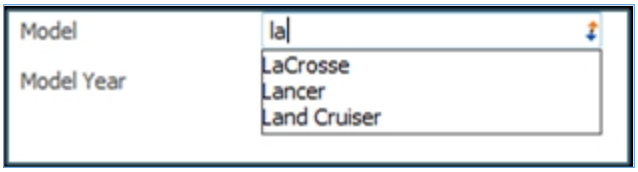
To help users complete input text fields on a form quickly and accurately, your application can present the user with a list of choices, from which one value can be selected with the mouse or keyboard. This reduces typing and helps ensure that values are spelled correctly when the form is submitted.

The AutoComplete and Dynamic Select features are two features that each support this capability. While similar, each has advantages in specific situations.

If an input text field always has fixed and short list of allowable values (say S, M, L, XL, XXL for sizes), neither of these controls is appropriate; the user can select from a short, static list of choices.

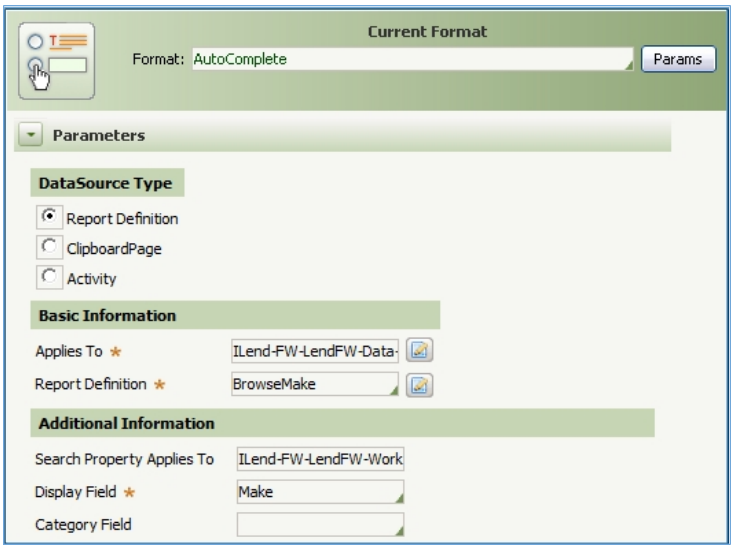
AutoComplete

An autocomplete input field on a form supports user selection of a text value from a possibly large list of candidate values. After the user types one or a few characters, a filtered list of matching text values appears below the input field. The user can select a value from the drop-down list. On an input field form, the icon indicates that autocomplete is available for the field.



AutoComplete controls are useful in cases when a list is generated from a report or an activity, the list is too long to be easily searched from a drop-down list, and the user is likely to know one or a few letters of the right value.

To add an AutoComplete control to a section rule, select and drag the icon  from the Advanced palette into a cell of a layout. Complete the parameters for the control:



Dynamic Select

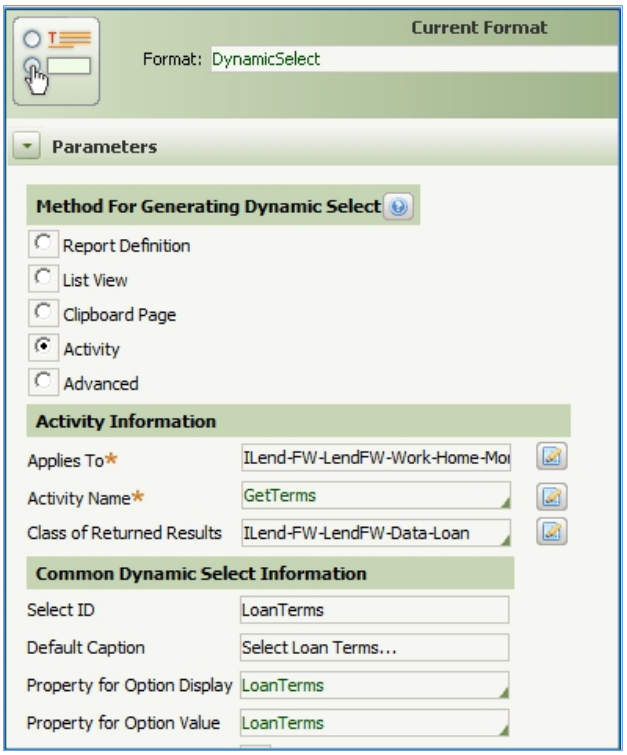
A Dynamic Select control displays a drop-down list of text value choices for an input field, returned from an activity, a list view, or a report definition. You can control the behavior using many different parameters and can link — or **cascade** — two or more Dynamic Select controls in sequence to refine lists as users make choices. For example, in a shopping application, users could first select a department, then a product, then a color or size for the product, using three cascading Dynamic Select controls

Dynamic select controls are best used when options for the drop down list are not static (may change depending on context or from time to time) and the list is not short. The list of choices can appear automatically when the user tabs into the input text field; no typing is necessary.

To generate a Dynamic Select list, use the following:

- **List view** — Use when the values that users select and the values for the property are both present as columns in a list view rule.
- **Report definition** — Use when the values that users select and the values for the property are both present as columns in a report definition rule.
- **Clipboard page** — Use when the values that users select and the values for the property are both present on a clipboard page.
- **Activity** — Use when there is an activity that produces an XML document needed by the Dynamic Select control.

To add a Dynamic Select control on a section rule, select and drag the icon  from the Advanced palette into a cell of a layout. Configure the parameters for the control:



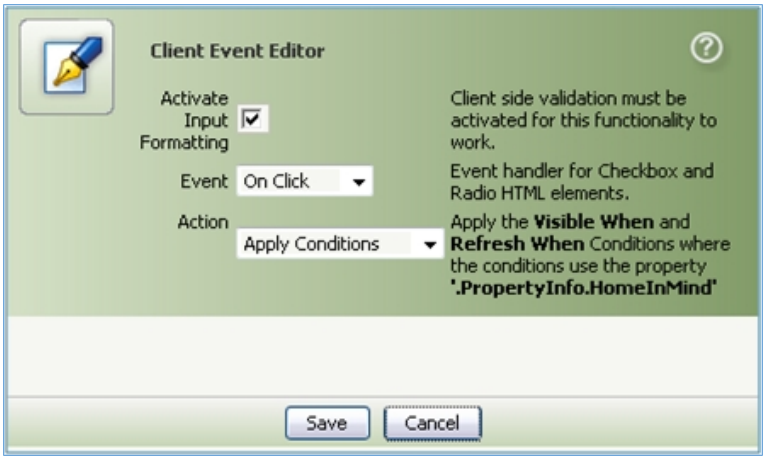
For more information on the parameters, see the PRPC Developer Help for Adding a Dynamic Select control.

Client Events

Client events render HTML rules when a user performs an action on screen. You define logic using a form and the Client Event generates JavaScript automatically. You may need to refresh entire section to display conditionally visible elements or new clipboard data.

Client Event Editor

The Client Event editor includes several events and actions.



Events include:

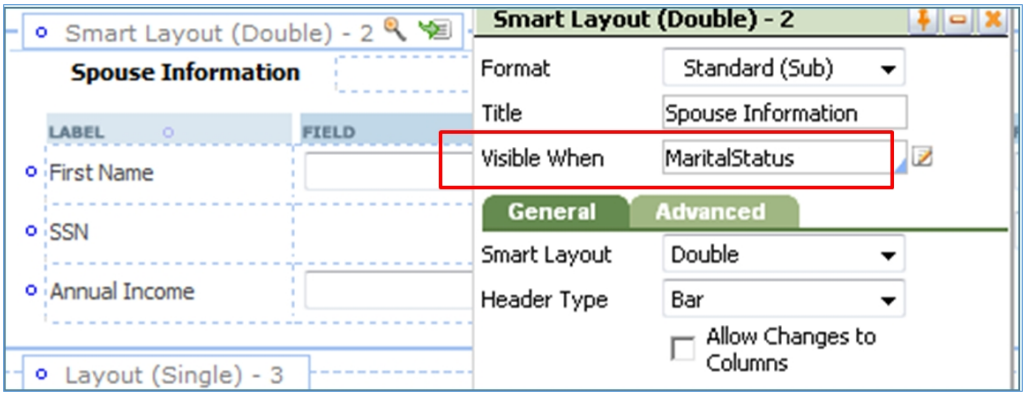
- On Change – useful for text fields
- On Blur – when focus shifts away from a control
- On Click – best for buttons and checkboxes

Actions include:

- Apply Conditions such as Visible When and Refresh When
- Refresh section
- Call activity

Client Events: Visible When

To trigger the client action, select a when rule in the Visible When field. Visible When fields are available in layouts, sections, fields, and labels.



Reference Material

PRPC Developer Help topic — Adding an AutoComplete control

Lesson_UI_Gallery

Objectives

At the end of this lesson, you will be able to:

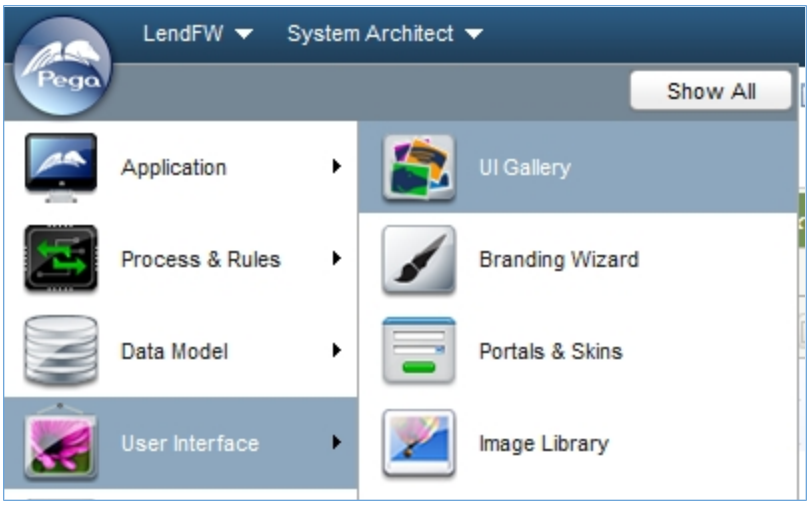
- Find and explore the UI Gallery
- Copy example controls
- Add custom samples to the UI Gallery

Things to Know

PRPC provides a vast array of user interface configurations which enables us to manipulate and adapt our applications to meet our customer's specific needs. With such a wide variety of options available however, it can be daunting to determine which tools to use for which tasks.

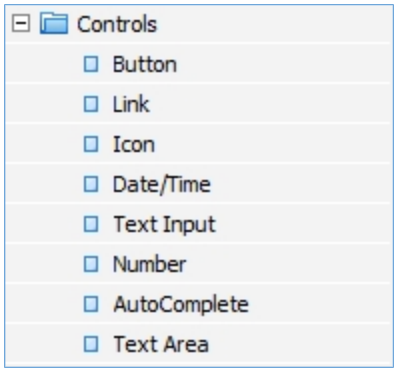
Find and explore the UI Gallery

The UI Gallery, accessed through the User Interface category under the Pega button, provides examples of effective and dynamic user interface configurations using many of Process Commander's design capabilities. The forms and layouts represent best practice implementations and are guardrail compliant.

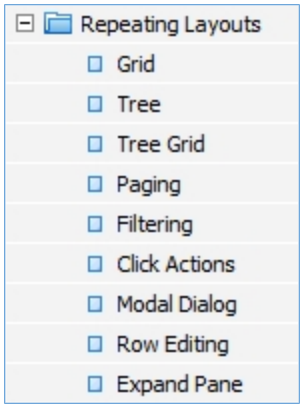


The UI Gallery is divided into four categories of configuration tools:

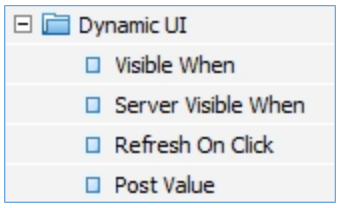
- **Controls**- demonstrates the features that control how properties appear on user forms, correspondence, and other HTML forms, for both display and for accepting user input.



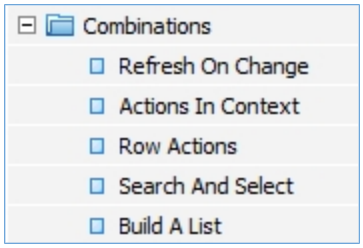
- **Repeating Layouts** - demonstrates the types and features of repeating layouts, which consist of one header row that contains property labels, and a row of property fields. These types of layouts are covered more extensively in the Repeating Layouts lesson of this course.



- **Dynamic UI**- demonstrates how to use user interface events such as entering a property value in an input box, or selecting a value from a Dynamic Select field, to automatically trigger UI actions such as refreshing a section, hiding a property or section, expanding a layout, and so on.

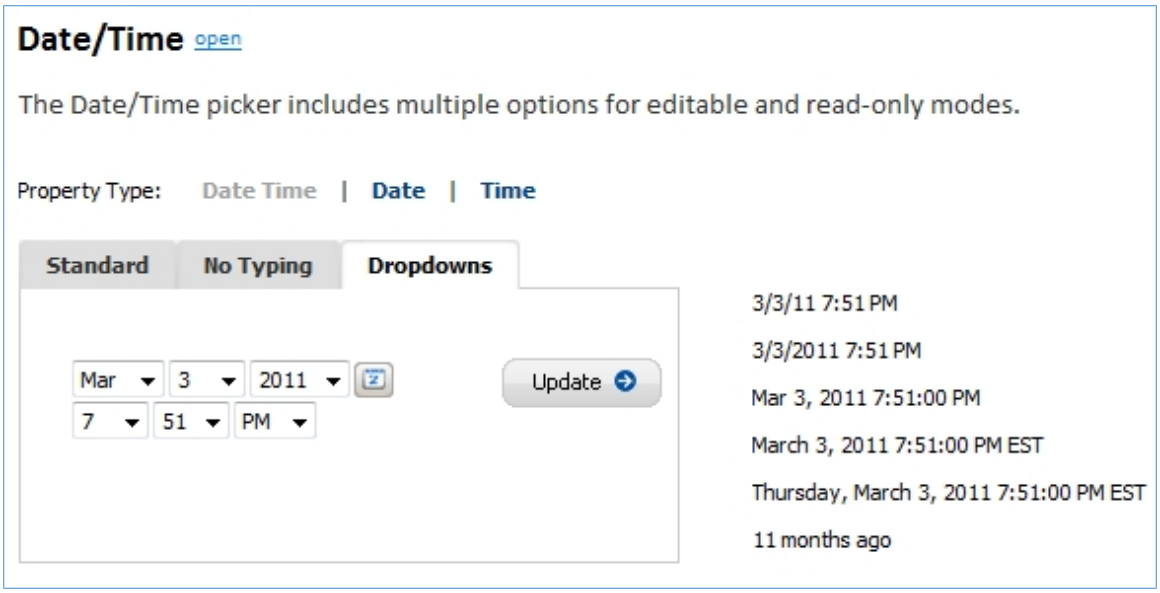


- **Combinations**- demonstrates how to use PRPC's UI configuration tools in concert to provide more powerful, straightforward user experiences.



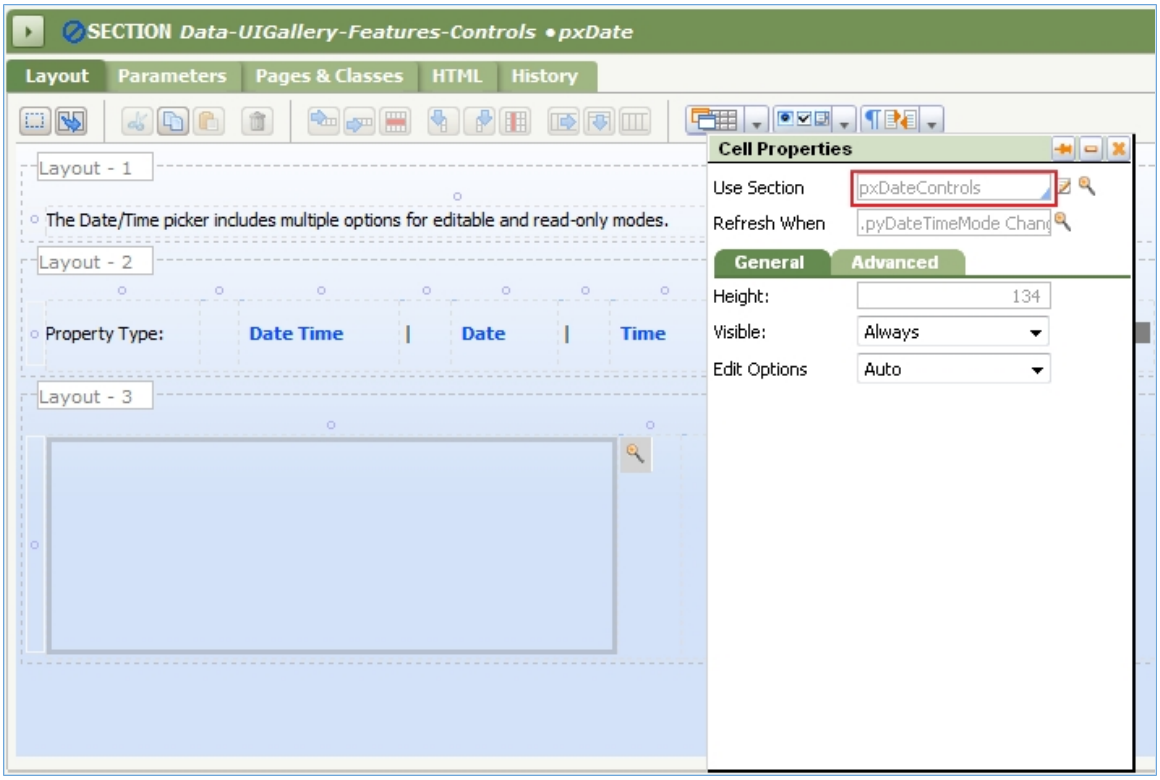
Copy example controls

Within each category there are dozens of helpful examples of how to best use the UI configuration tools. Let's take a look at one example.

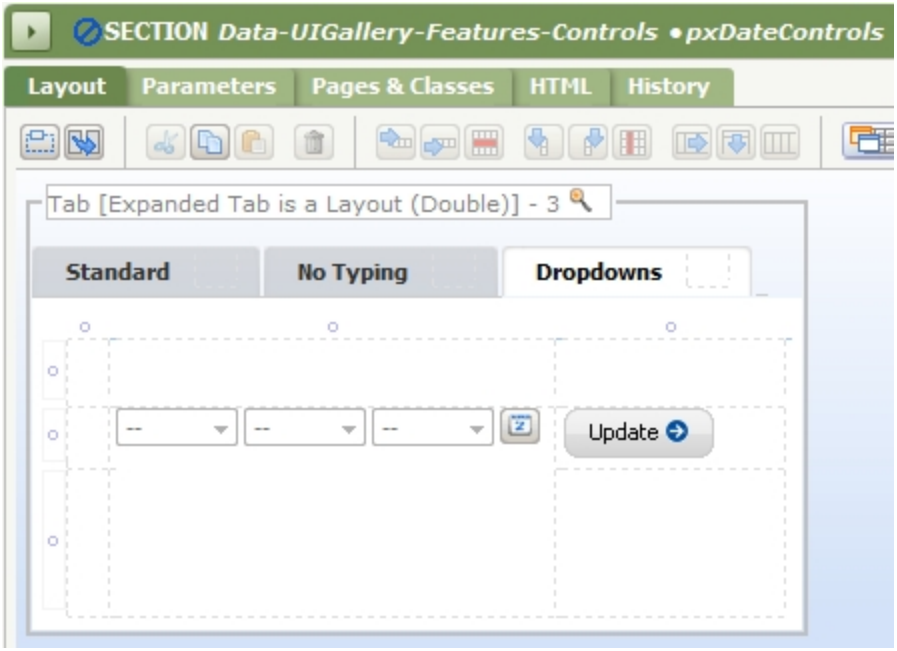


Here we can see several different ways to configure the look and feel of date input controls. If we wanted to use one of the samples to configure our own UI, we simply click the open link at the top of the form to open the underlying rules.

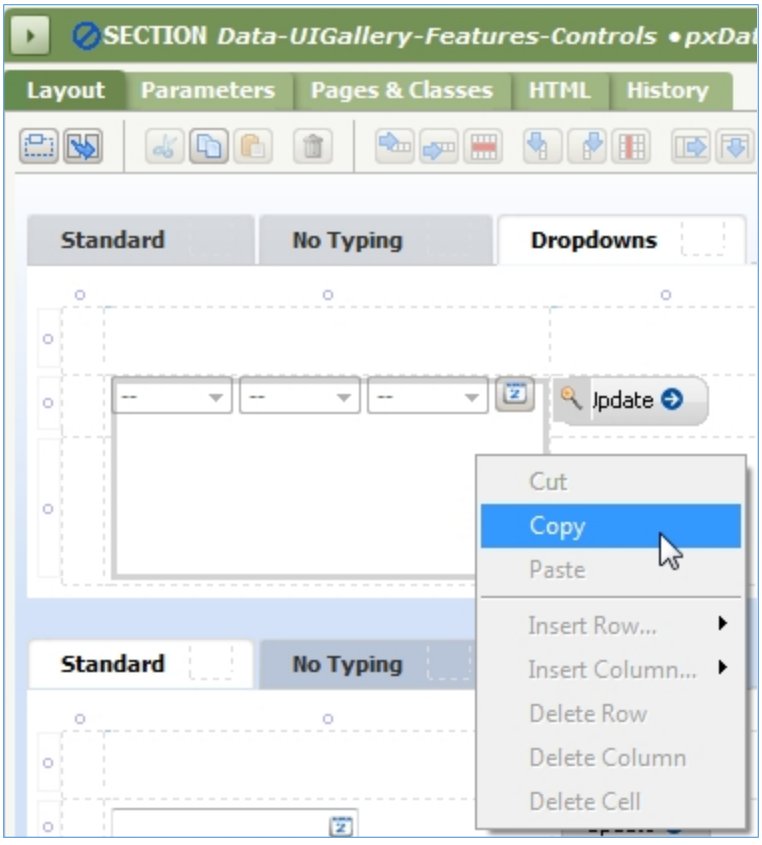
Clicking open in this case reveals the pxDate section, which contains several sections within it.

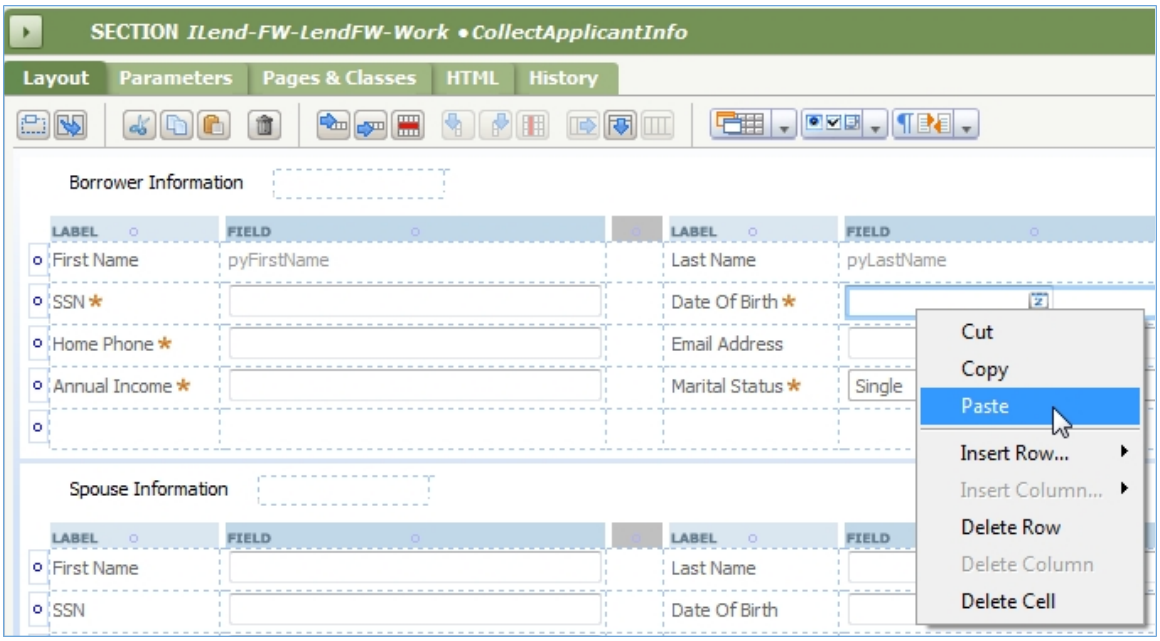


In particular, in this case we are interested in the pxDateControls section, which contains the sample dropdown date controls for us to copy into our own section.

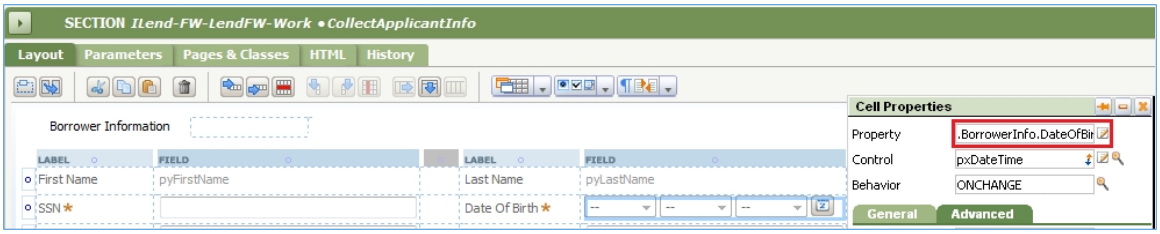


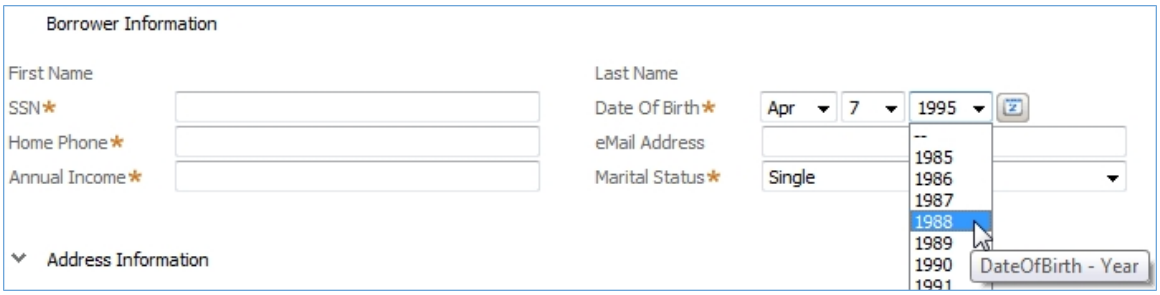
From here, we simply right-click the relevant sample, click copy, and paste it into our section.





Finally, we ensure the correct property is referenced and preview the section to see the new control.





Module 15: Validation

Lessons Covered:

- Validation Basics
- Validation Rules

Validation Basics

Objectives

At the end of this lesson, you will be able to:

- Describe validation of data
- Identify the different ways used to validate incoming data in your application
- Use field validation to prevent empty user input values

Things to Know

Validation ensures the quality of the information involved in the application and its business process. In general, validation of data involves examining data values to ensure that they meet the requirements of the application. During validation, the data values coming into the application from a source (users, files, or external systems) are compared against defined criteria. If the incoming information does not meet the criteria, an error is reported to the system. Depending on the requirements of the application, the system can take further action, or reject the incoming data entirely.

Typically a corrective action is requested. For example, if an input field requires a value of a certain length (such as a ten-digit telephone number) and the user enters too few digits, the system prompts the user to review and correct the input.

Comprehensive validation of user inputs improves the quality of your application and increases application security, by catching missing or incorrect data as early as possible.

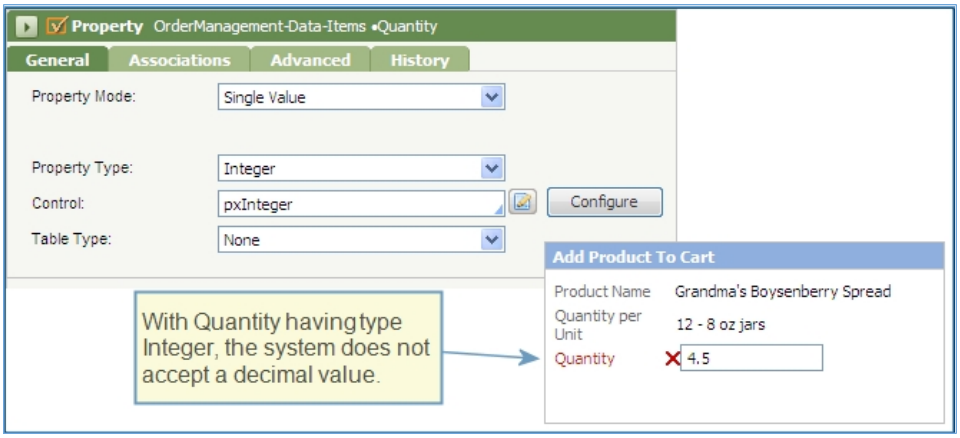
Typical Ways to Validate Incoming Data in an Application

The following items are typical ways to validate data coming into your application, and constrain the data to allowable values.

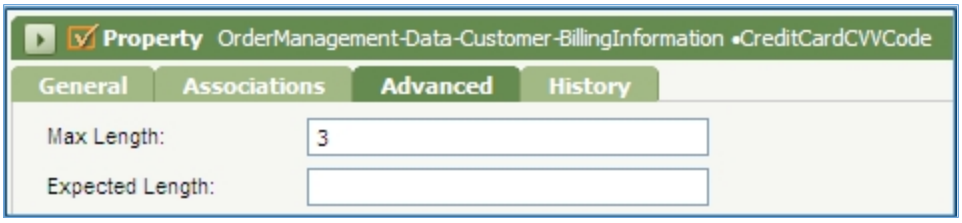
Constraining Input Values By Property Characteristics

Constrain input values for a specific property by using settings such as the property type and maximum lengthsettings. When a data value is entered into a field corresponding to that property, the system automatically validates the input against the rule form's settings.

For example, when the property type is defined as Integer, a decimal value is not allowed.

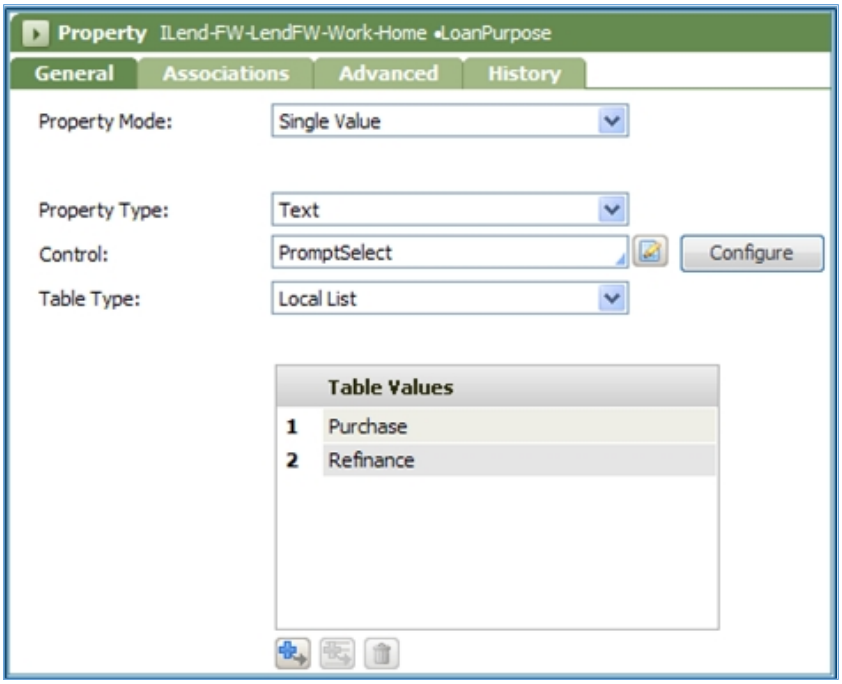


For example, when a text property has a maximum length set to three (3), the user cannot enter more than three characters into the field.



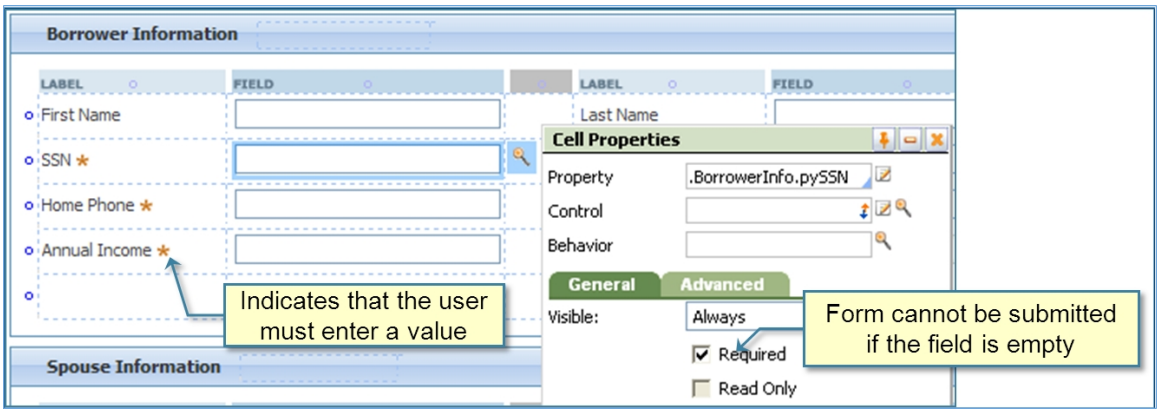
Constraining Values By Limiting To A List

A Local List defines a list of acceptable values for a particular property. At runtime, the system compares the input values to the specified set and verifies whether the value has a match among the listed choices. Using a Local List is the simplest way to show a list to the user at runtime for the user to make a selection.



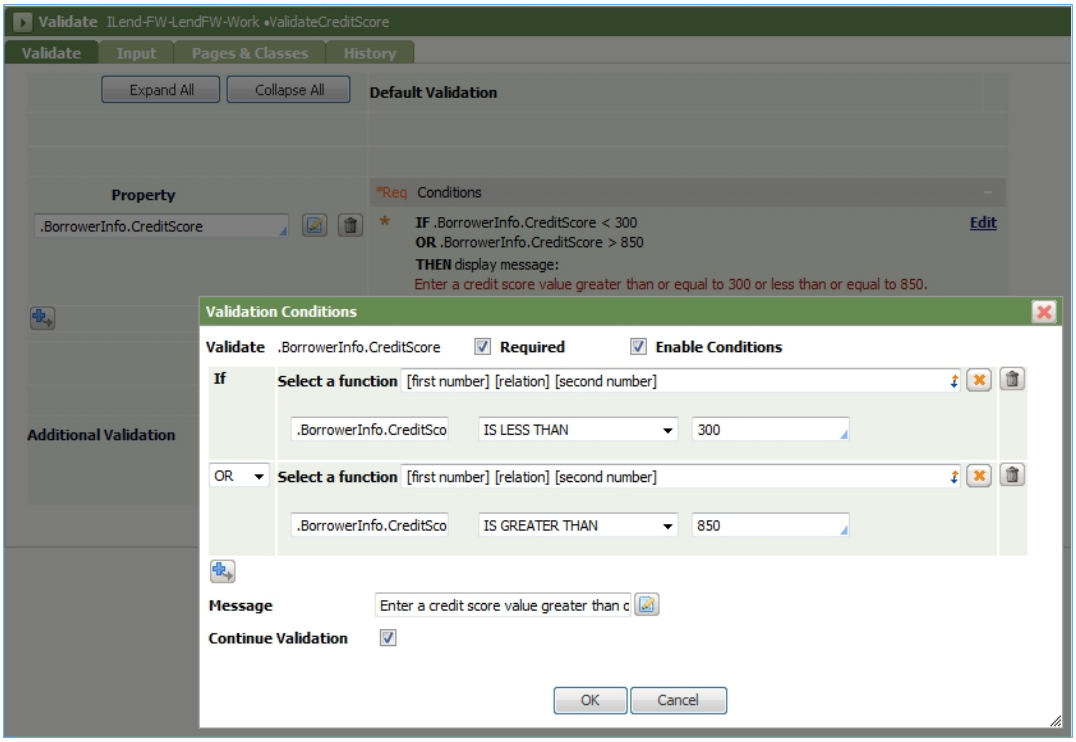
Requiring Input

To ensure users fill in a required field on a form select the **Required** checkbox in the Cell Properties for that entry field. The system automatically adds a visual indicator to the label to alert users that the field is required. If users do not enter a value and try to submit the form, the system displays an alert.



Expression-Based Validation

Validate rules provide the ability to do sophisticated validation and present custom error messages. These rules validate data entered by users or received from another system or source. They can be explicitly called by a flow action or an activity. A validate rule can test multiple input values at once.



Advanced Validation

Edit validate rules provide more sophisticated validation and custom error messages than the validation available solely using the Property rule form, the layout field, or validate rules. However, these rules require Java programming skills to develop, because they use a Java function to test the validity of a single input. They can be referenced by property rules and called by validate rules and by activities (using the Property-Validate method).

Comparison of the Validation Methods

The following table describes where each is enforced by the system.

Validation Method	Where Set	Where Enforced
Property type	Property rule form	Wherever that property is used in the application, regardless of which section the property is used in.
Limiting to a list	Property rule form	Wherever that property is used in the application, regardless of which section the property is used in.
Requiring a value	Section rule form (in the Cell Properties for the value)	In that specific section.
Validate rule	Called explicitly	From the specific flow action or activity that calls the validate rule. Helpful when you want various properties on different flow actions to trigger the same validation test. For example, the UpdateIncome and UpdateLoanAmount flow actions might both call the same validate rule to test if the loan should be approved.
Edit validate rule	Property rule form Called explicitly	If specified on the property rule form, the validation is enforced wherever that property is used in the application. If called explicitly, from the specific validate rule or activity that calls the edit validate rule. Helpful when different properties or different validate rules or activities might reuse the same validation test.

Validation Rules

Objectives

At the end of this lesson, you will be able to:

- Identify the key differences between the validate rule and the edit validate rule
- Identify the key elements of a validate rule
- Describe how to associate a validate rule with a proposed work status
- Describe how to use input-qualified validation
- Use a validate rule to validate user input data entered into the user interface for a flow action

Things to Know

Differences between Validate Rules and Edit Validate Rules

Both the validate rule and the edit validate rule can test the validity of an input value. The following table lists the key differences between the validate rule and the edit validate rule.

	Validate Rule	Edit Validate Rule
Any special skills required to define the valid conditions?	No special skills needed. Expression-based.	Yes, requires Java programming skills.
Allows for presenting custom error messages to the user?	Yes	Yes
Which rule types usually call it?	Flow action rules, activities	Property rules, validate rules, activities (in a Property-Validate method)
Can it test multiple input property values in a single rule?	Yes, a validate rule can test multiple property values	No, each edit validate rule tests a single input value
In which Application Explorer category does it appear?	Process	Data Model

In general, it is best to use validate rules instead of edit validate rules because validate rules:

- Are easier for non-programmers to design and understand.
- Are more connected with the business process and the user interface, as flow actions can call them.
- Simplify application design, because one validate rule can verify multiple input values at once (such as the multiple values involved in the calling flow action).

Note: Since edit validate rules require Java programming skills, they are outside the scope of this course.

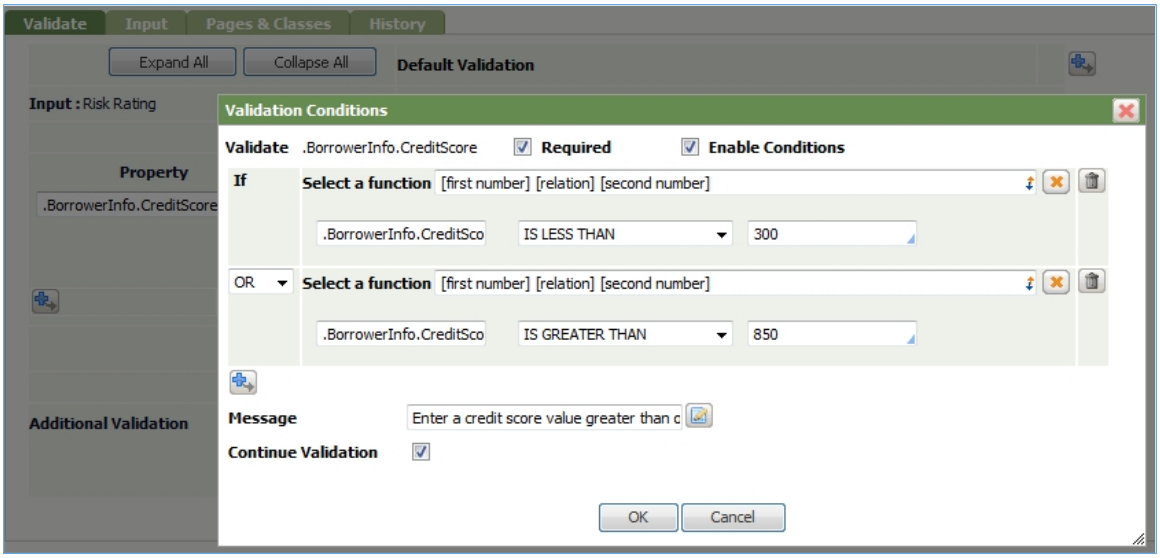
Key Elements of a Validate Rule

The most important tab is the Validate tab. The key elements to specify in the Validate tab are:

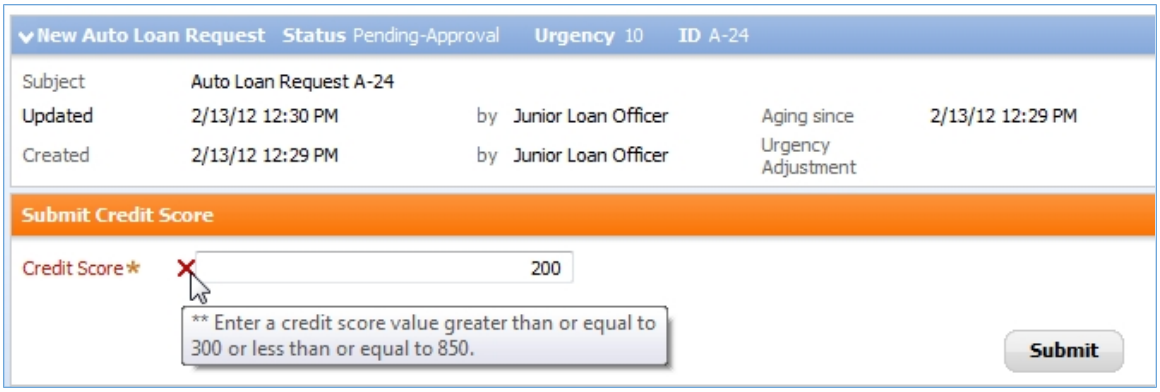
- The **property** to be tested
- Whether the user is **required** to enter a value for that property

- The **conditions** that the property's input value must meet
- Whether the system is to **continue evaluating** other conditions if the value fails to meet an earlier condition

In this example, the validate rule tests the CreditScore property, and requires users to enter a value that meets certain conditions. The condition tests whether the value entered for ScreditScore does not fall within the 300 to 850 range. If it does not match, upon submitting the form, the validation fails and the message "Enter a credit score value greater than or equal to 300 or less than or equal to 850." displays to the user.



The screen shot below shows the message that displays in the user form when the user submits a non-matching value.



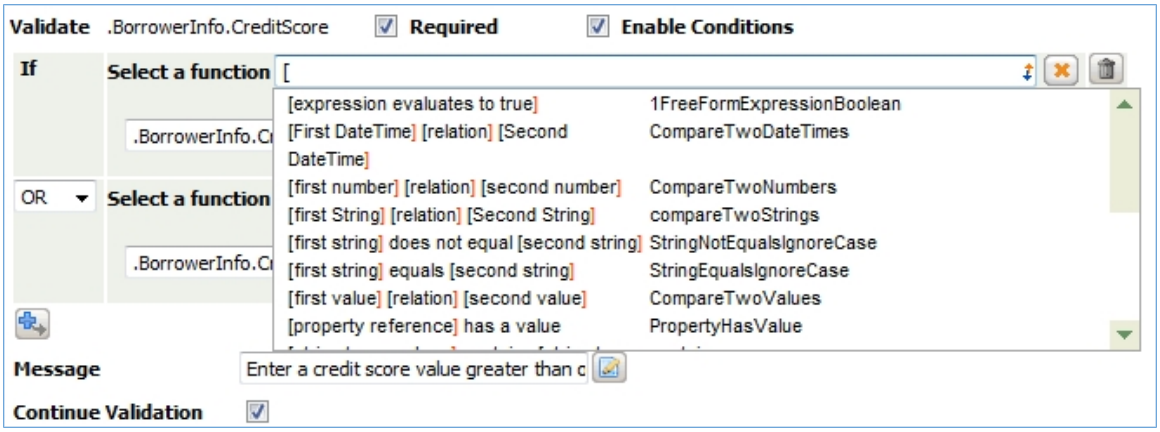
Specifying Validation Conditions

To specify the conditions for a property, click the **Edit Advanced Options** ( ) icon in the **Details** column for that property, and set the criteria in the Validation Conditions window that opens.

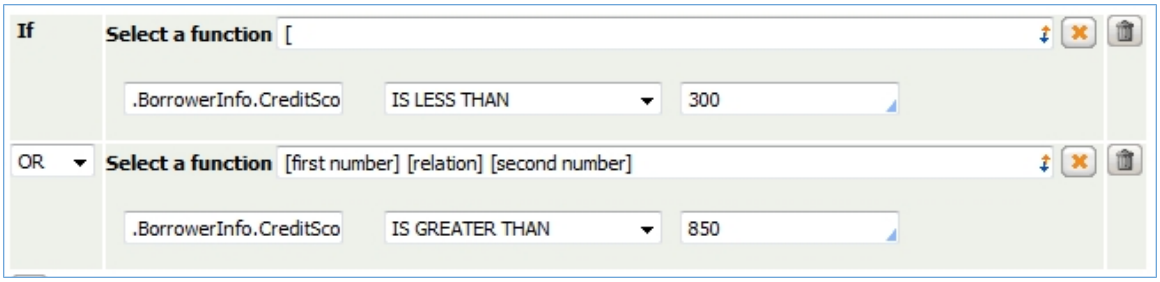
Note: When setting detailed conditions using the Validation Conditions window, the system automatically sets the appropriate validate option (such as **Validate** or **Validate Each**) based on the property mode of the property to be tested, and displays it at the top of the Validation Conditions window.

We set the conditions to check whether the user has entered an undesired value. In a sense, the validation is looking for *failure*. When the system tests the input value, if the specified conditions are all met, the input value fails validation.

Use the AutoComplete (*) to select an expression type for the **If statement**. To use the autocomplete, we set the cursor in the field and press the down arrow on the keyboard. If there is an expression already in the field, we might have to delete it and then use the autocomplete to see the full list of available expressions.



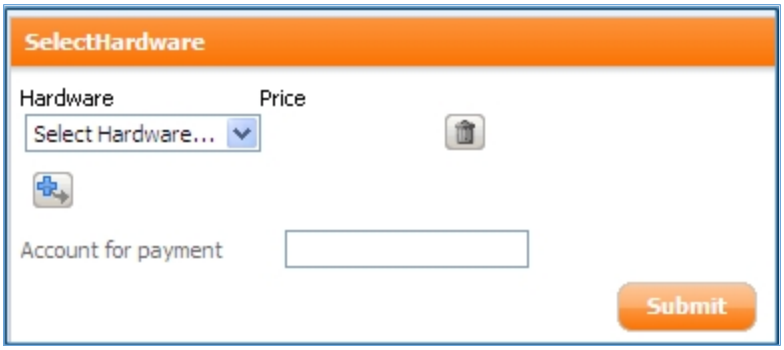
In the Validation Conditions window, click  to add more conditions to the **If statement**, to specify criteria such as "If x OR y OR z and "If x AND y OR z" on the input value. Additional conditions are shown in the screen shot below.



Calling a Validate Rule from a Flow Action

We specify a validate rule in a flow action rule form to validate the values that users enter in the form that is associated with that flow action. The input values are tested when users click the **Submit** button.

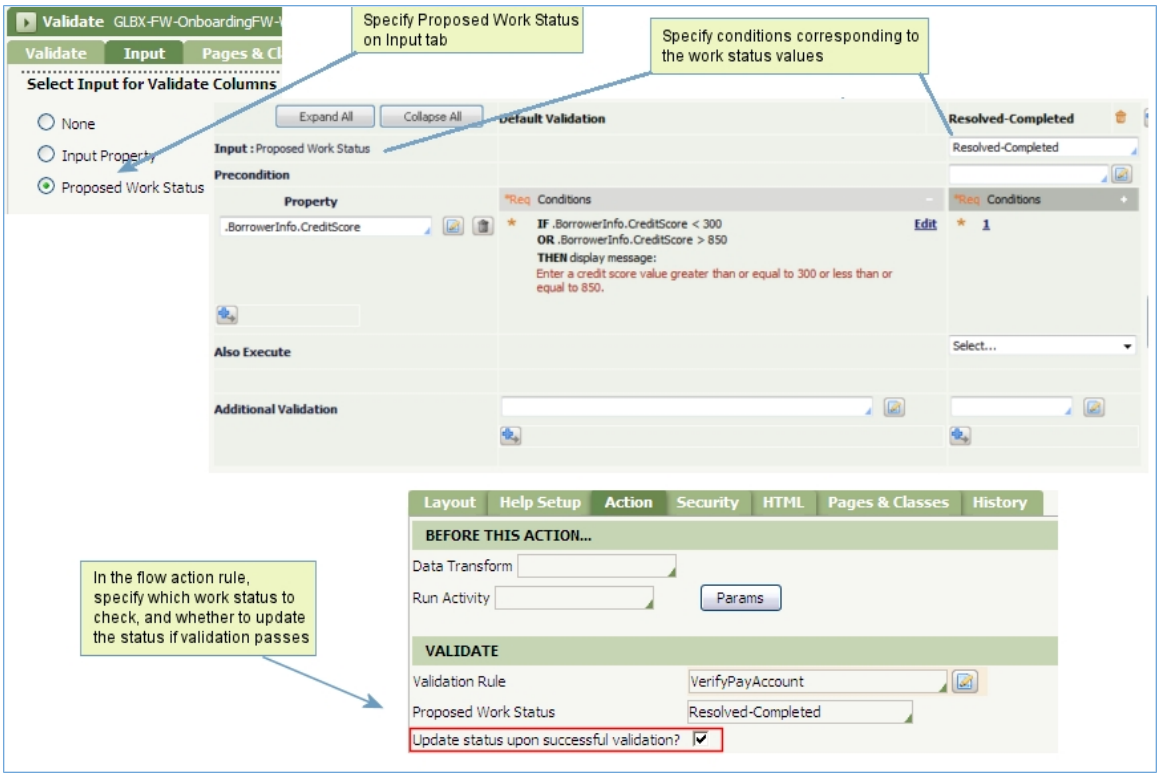
For example, the SelectHardware flow action displays the following form to users:



The Action tab of the SelectHardware flow action specifies the VerifyPayAccount validate rule.

Setting the Work Status after Passing Validation

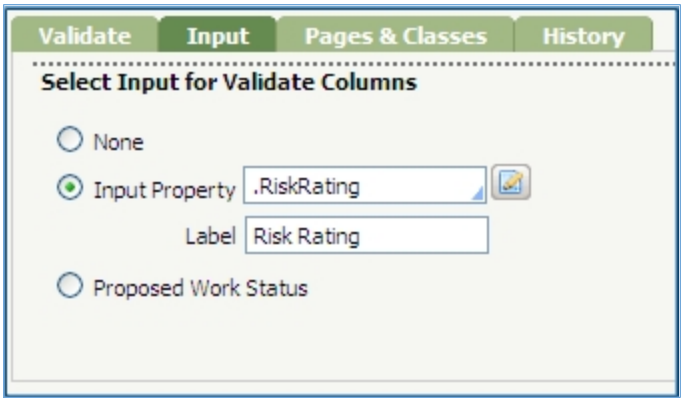
A powerful way to use a validate rule in a flow action is to have the system automatically update the work status of the work item when the input values pass the validation conditions required for that work status. For example, if the work item cannot be considered Resolved-Complete unless the input values meet certain criteria, it can specify validation conditions related to that specific proposed work status. Then when the input values pass the criteria associated with that work status, the work can be automatically resolved without further human intervention.



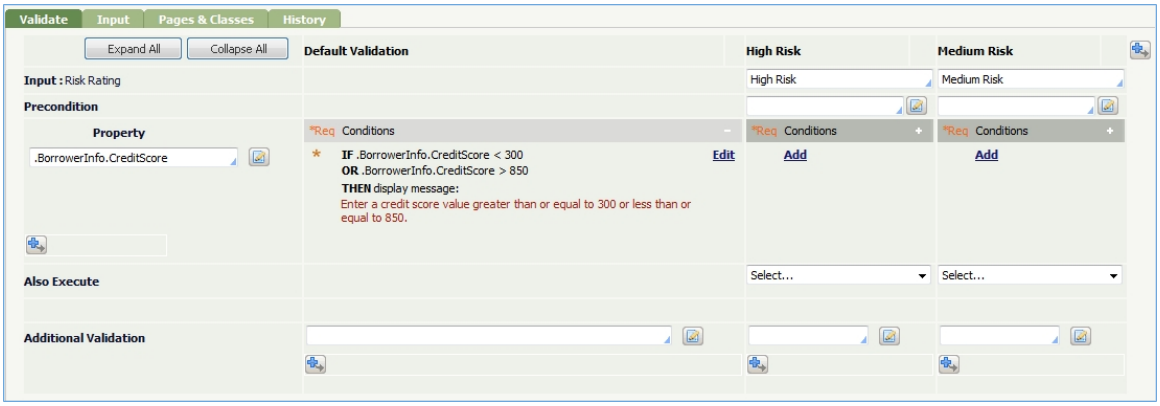
Input-Qualified Validation

Similar to defining specific validation conditions for particular work status values, it can associate different validation conditions with specific input properties within a single validate rule.

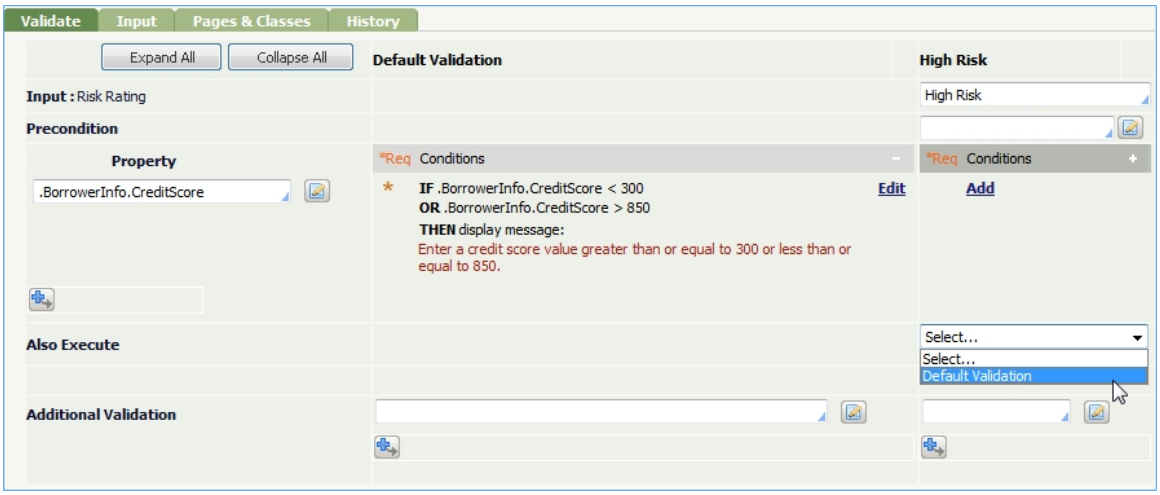
For example, in a loan application, we might want to have different validation criteria depending on the loan applicant's risk rating (medium or high). Given an input property RiskRating that has a **Local List Table Type** with values High Risk, Medium Risk, and Low Risk, in the validate rule form, we can define one set of validation conditions for High Risk and another set for Medium Risk, by specifying an input property for the validate rule.



When the input property is specified for the validate rule, we can define the sets of validation conditions that correspond to each of the input property's values.



When using input-qualified validation, the system locates the column corresponding to the value of the input property, and verifies just the conditions from that column, unless we make a selection in the **Also Execute** drop-down list, including the default validation. That is, we must explicitly select whether to also run the conditions in the **Default Validation** column by selecting it in the **Also Execute** drop-down list. Otherwise, the system only checks conditions corresponding to the particular value of the input property.


Validate
Input
Pages & Classes
History
Expand All
Collapse All
Default Validation
High Risk
Input : Risk Rating
High Risk
Precondition
Property
*Req Conditions
*Req Conditions
.BorrowerInfo.CreditScore
IF .BorrowerInfo.CreditScore < 300
OR .BorrowerInfo.CreditScore > 850
THEN display message:
Enter a credit score value greater than or equal to 300 or less than or equal to 850.
Edit
Add
Also Execute
Select...
Select...
Default Validation
Additional Validation

Module 16: Creating Correspondence

Lessons Covered:

- Correspondence Basics
- Work Parties
- Correspondence Activities

Correspondence Basics

Objectives

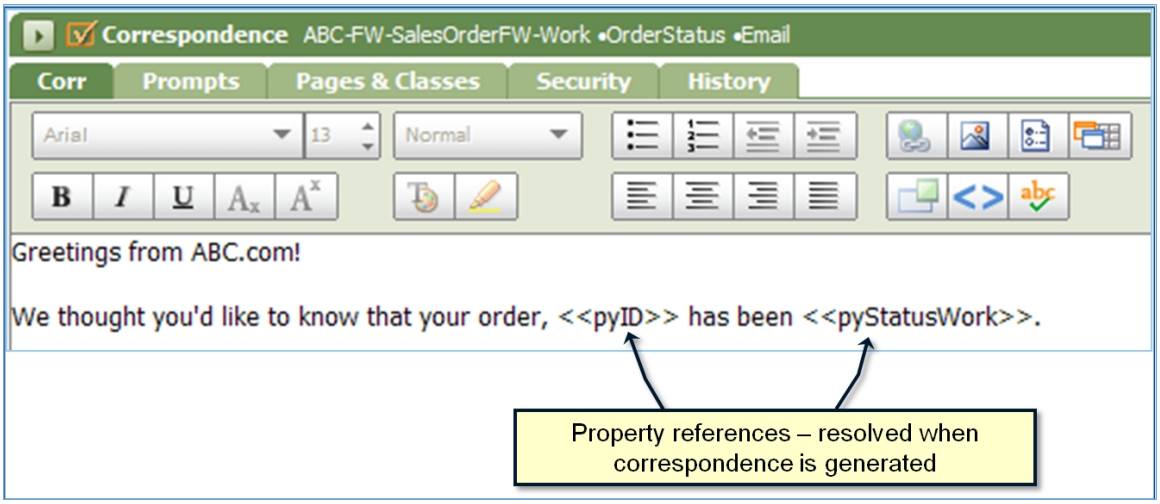
At the end of this lesson, you will be able to:

- Use correspondence rules
- Use the Rich-Text Editor
- Describe correspondence types
- Create correspondence Rules
- Send correspondence using assignment shapes

Things to Know

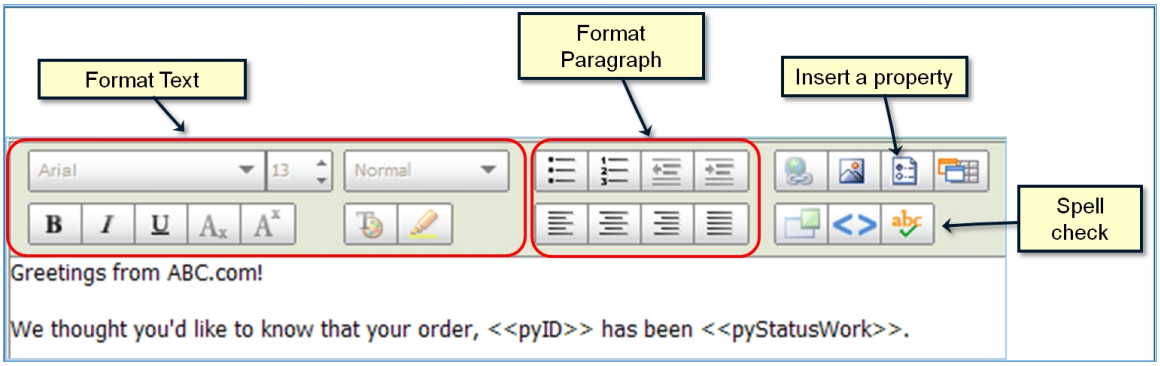
Correspondence Rules

Correspondence rules are templates that we can use to create outgoing notifications. They can include properties and other rules for customized output and are located in the Process category.



Using the Rich-Text Editor

The rich-text editor allows us to create formatted, professional-looking correspondence or paragraph rules.

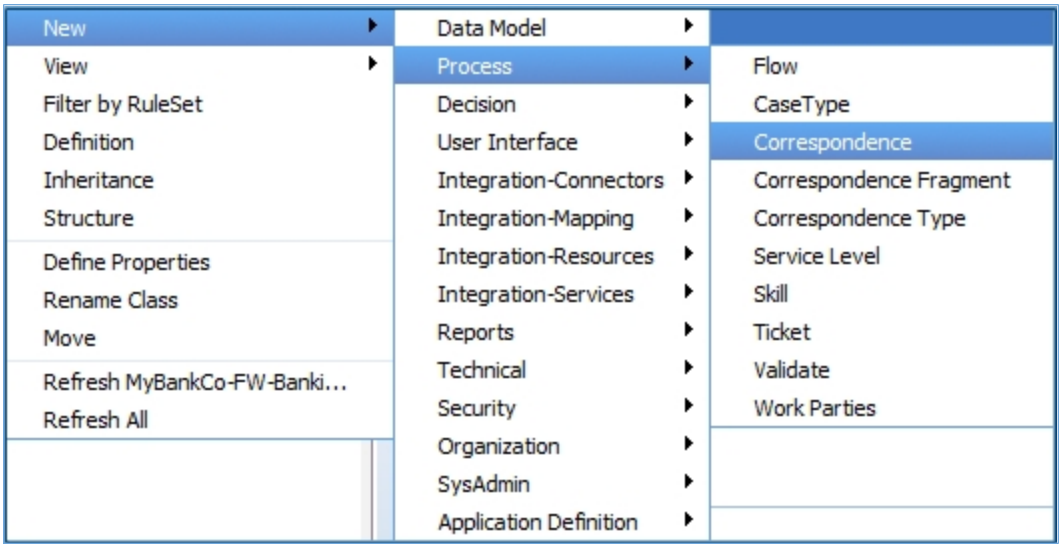


Correspondence Types

There are four standard correspondence types: email, fax, mail, and SMS Text. Correspondence is associated with work parties.

Creating Correspondence Rules

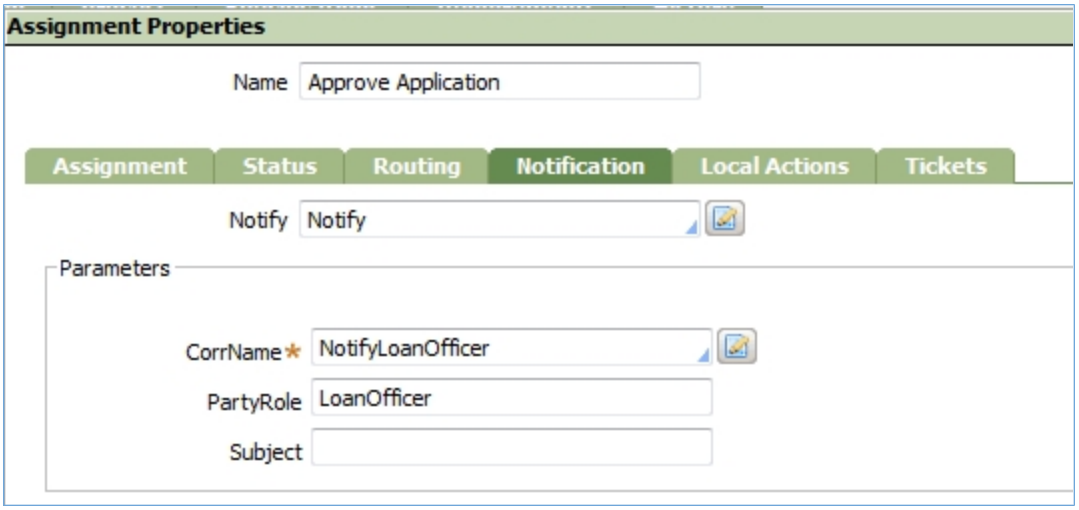
We can create a new piece of correspondence by selecting the rule type from the Process category.



Correspondence rules can also be automatically generated when we run the Application accelerator.

Sending Correspondence Using Assignment Shapes

Use the Notify tab of an Assignment shape to automatically send a notification message. Specify the activity in the Notify field. This will populate the Parameters area, where you can then specify which correspondence rule to use.



Work Parties

Objectives

At the end of this lesson, you will be able to:

- Describe a Work Party, Work Party Roles, and Party Classes
- Define Data Party Classes
- Describe Work Parties and the Clipboard
- Configure a Work Parties Rule
- Add a Work Party from Assignments
- Route to Work Parties

Things to Know

What is a Work Party?

A work party is a person, organization, or other actor identified in a work item, who needs to be notified or is interested in the outcome of a work item or case. A work party can be the recipient of email or other forms of correspondence regarding the progress (status) of a work item. In addition to correspondence, we can use work parties for user interface display and reporting.

For example, we can specify that the creator of a work item is the Customer. This enables the application to automatically notify the Customer when their work item has been approved, using the email address from the work parties' operator profile.

The work party mechanism makes it easy to capture and store information about a party, person, or other entity that may be associated with a work item. PRPC provides several standard work parties, such as Originator (the default work party), Customer, and Owner; many of these parties are used in various standard reports. A work item does not have to have any work party associated with it. However, if it does, there may be one, a few, or a large number.

Some of the associated work parties may not have any responsibility in the processing of the work item. For example, a lender in an external organization may be interested in the outcome but has no responsibility in the processing, or a manager may be interested in how the work item gets resolved, but is not responsible for its processing.

Work Party Roles

A work party role identifies why a party is participating and their involvement in the process. The role may also determine the properties that are defined for that work party. Work party information is aggregated by party roles, such as Originator (a required role, identifying the party who directly caused the work item to exist), Customer, or Manager.

Some work parties may be categorized as "interested parties"; this typically means that they do not process a work item, but are interested in its progress and can receive correspondence as the work item progresses to completion. For example, a car loan is signed only by the husband in a family; his role is "Borrower" and he is a participant in the work item's processing. His wife may play no legal role in the car loan, but may want to receive email about its status and progress; she is only an "interested" party.

A work parties rule is referenced on the Process tab of a flow rule. If the Work parties field on that tab is blank, PRPC automatically looks for any applicable work parties rule named Default (work party rules are named Default by convention).

Work Parties and Party Classes

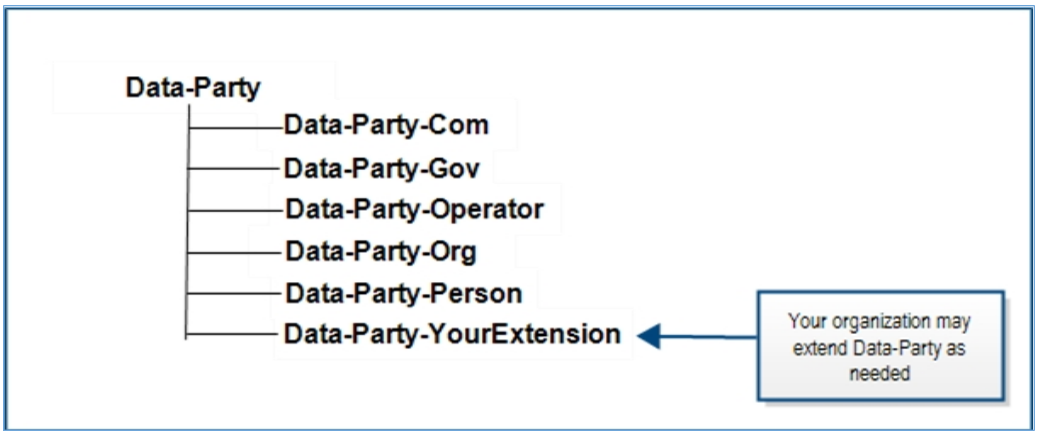
Work parties use the following standard data party classes to hold commonly used properties for work party roles.

- Data-Party-Com — for business organizations
- Data-Party-Gov — for government organizations
- Data-Party-Operator — for Process Commander operators
- Data-Party-Org — for non-profit organizations
- Data-Party-Person — for individuals

Data Party Classes

PRPC provides many standard, pre-built properties, such as First Name, Last Name, and Email Address that are defined in Data-Party classes. These classes also contain pre-built sections that are useful and reusable.

Work parties are represented as a repeating group (a Page Group) within a work item. We can use any of the standard subclasses of Data-Party, or create a custom subclass if needed.

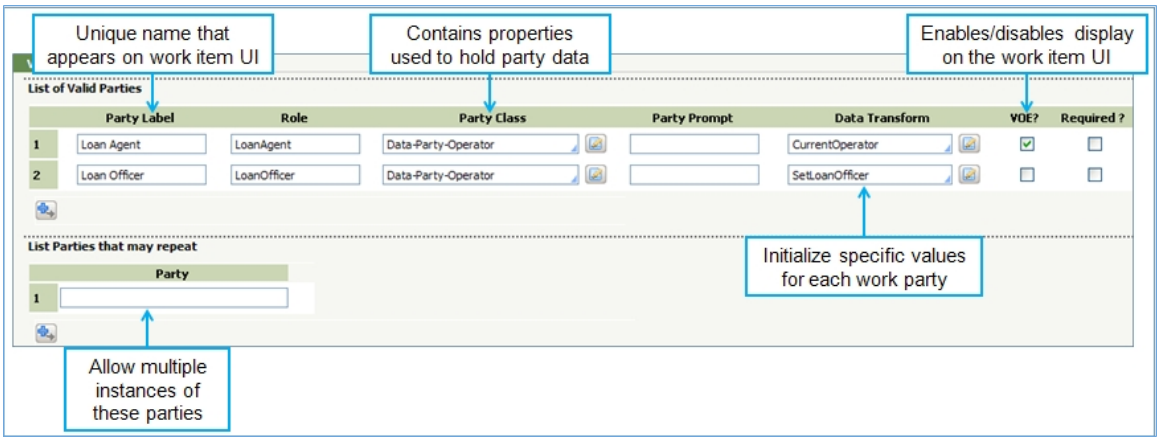


Configuring a Work Parties Rule

A work parties rule defines which work party roles can participate in a work item, and maps the role to a data class. Each work item can contain many roles in addition to the required originator role. Some roles may participate with multiple occurrences, for instance a role named "Child". The work parties rule also controls how and whether users can add a party using the user forms.

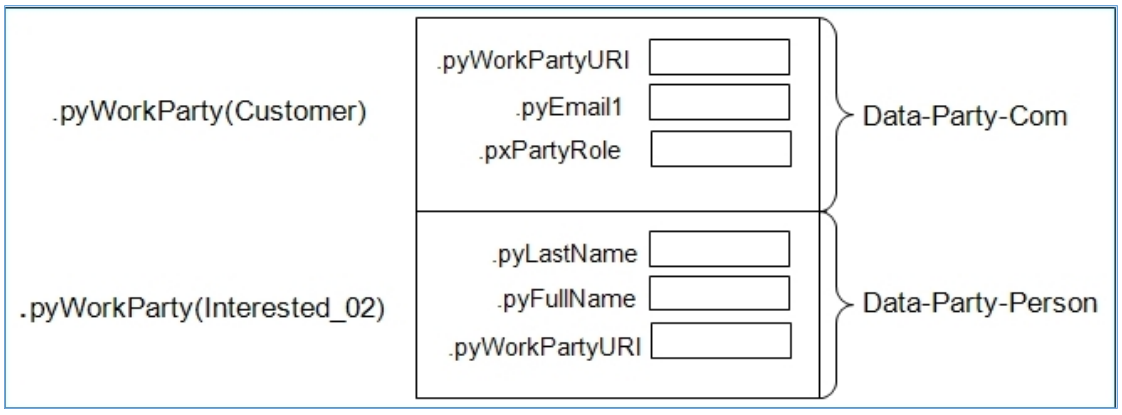
A work parties rule is referenced on the Process tab of a starter flow. The Application Accelerator automatically creates a work parties rule named Default, based on information we provide in the Roles step, and references the rule on any starter flow it creates.

If the Work parties field on a flow rule is left blank, PRPC automatically looks for any applicable work parties rule named Default (work party rules are named Default by convention).



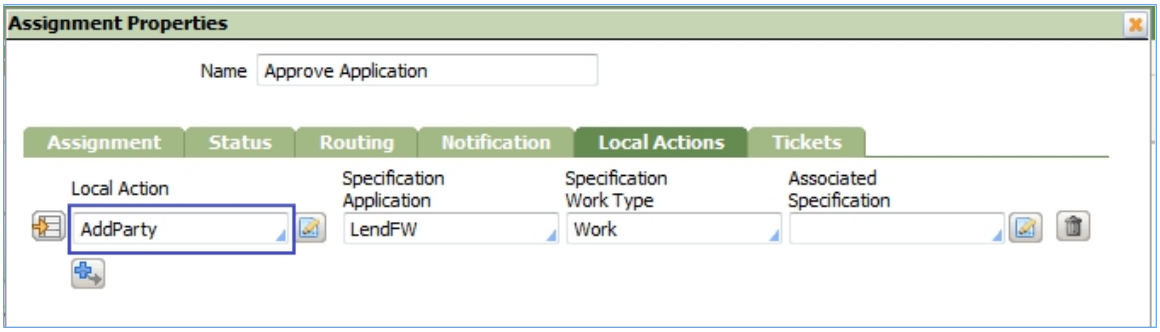
Work Parties and the Clipboard

When work parties are associated with a work item, a built-in property is embedded into the work item – .pyWorkParty. This property is defined as a Page Group of class Data-Party; it contains the details of each work party associated with a work item. The roles (indexes) are specified in .pyWorkParty.



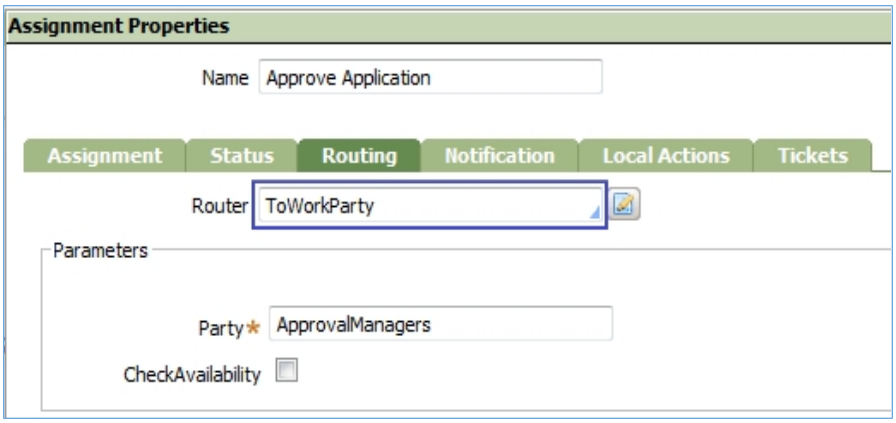
Adding a Work Party from Assignments

To allow users to add work parties while they process assignments, include the AddParty local flow action in the Assignment Properties panel.



Routing Assignments to Work Parties

We can use the standard routing activity **ToWorkParty** to send an assignment to the worklist of a specific work party of type Data-Party-Operator. For example, we can route the assignment to the user identified as the Originator, Manager, Underwriter, or QATester for a work item.


Assignment Properties
Name
Approve Application
Assignment
Status
Routing
Notification
Local Actions
Tickets
Router
ToWorkParty
Parameters
Party*
ApprovalManagers
CheckAvailability

Correspondence Activities

Objectives

At the end of this lesson, you will be able to:

- Define correspondence activities
- Understand when to use a utility shape and when to use the Notify tab on an assignment shape to send correspondence

Things to Know

Standard Correspondence Activities

A correspondence activity causes the system to send outgoing correspondence, such as an email message or a letter, to someone (or more than one recipient) about a work item. The message may simply inform the recipient that the work item has reached a certain stage, or may ask for information or a response.

PRPC includes several standard correspondence activities, including:

- **CorrNew** — used typically when generating correspondence from a utility
- **CorrQuickStart** — simpler correspondence generation activity, with fewer options than CorrNew

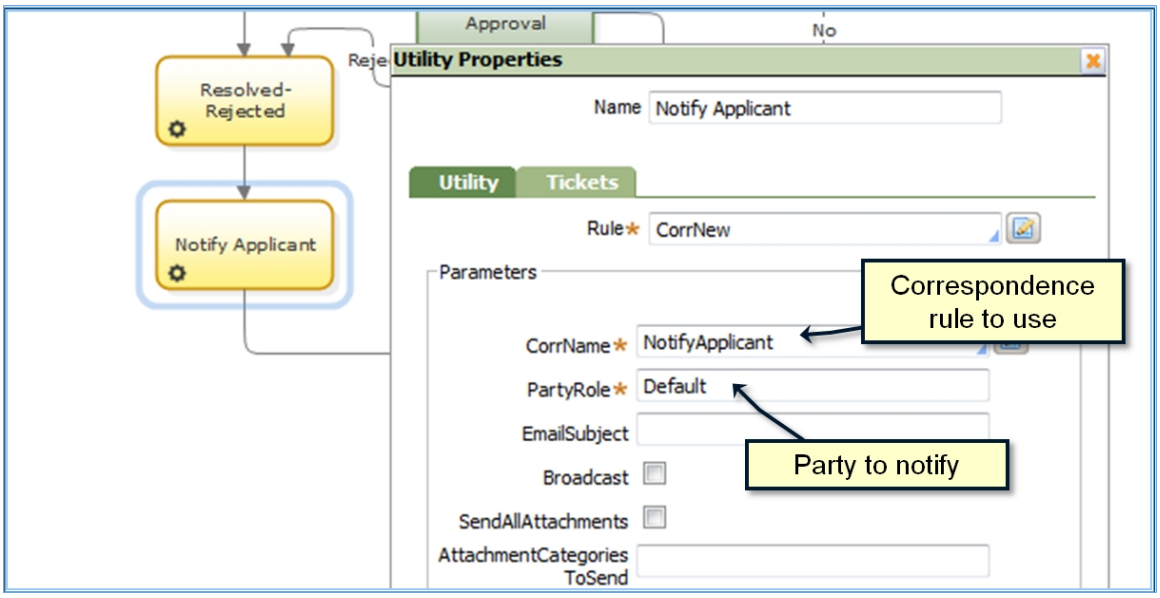
The following input parameters apply to these activities:

- **CorrName** (required) — name of the correspondence rule that defines the message contents
- **PartyRole** (required) — role of the person to whom the correspondence is to be sent
- **Broadcast** — correspondence is sent to all parties associated with the work item

When to Use a Utility Shape Rather Than an Assignment Shape

In a flow rule, you can use either of two shapes to generate correspondence:

- **Notify tab on Assignment shape** — Complete this tab to send an automatic notification message about new assignments and with standard notify activities (such as Work-.Notify). Notification messages typically are sent to the users who have the assignments, alerting them that more work has arrived on their worklist.
- **Utility shape** — Use to send any kind of correspondence programmatically to any party or parties, often with the standard CorrNew activity.


Approval
No
Resolved-Rejected
Notify Applicant
Utility Properties
Name
Notify Applicant
Utility
Tickets
Rule*
CorrNew
Parameters
CorrName*
NotifyApplicant
PartyRole*
Default
EmailSubject
Broadcast
SendAllAttachments
AttachmentCategories ToSend
Correspondence rule to use
Party to notify

Module 17: Guardrails and Best Practices

Lessons Covered:

- Ten Guardrails to Success
- Reviewing an Application
- Documenting Your Application

Ten Guardrails to Success

Objectives

At the end of this lesson, you will be able to:

- Describe and discuss the ten guardrails to success
- Identify when a guardrail has not been implemented

Things to Know

The following list describes the ten guardrails to PRPC success:

1. Adopt an Iterative Approach
2. Establish a Robust Foundation
3. Do Nothing That is Hard
4. Limit Custom Java
5. Build for Change®
6. Design Intent-driven Process
7. Create Easy-to-Read Flows
8. Monitor Performance Regularly
9. Calculate and Edit Declaratively, Not Procedurally
10. Keep Security Object-Oriented Too

1. Adopt an Iterative Approach

Define an initial project scope that can be delivered and provide business benefit within 60-90 days from design to implementation. Document five concrete use case scenarios up front and evaluate them at the end to quantify benefits, then use your scenarios as storyboards and ensure that each delivers a measurable business benefit.

2. Establish a Robust Foundation

Design your class structure to comply with the recommended class pattern — it should be understandable, easy to extend, and utilize the standard work and data classes appropriately. Use your organization entities as a starting point, and then proceed with class groups.

Lead with work items — create the class structure and “completed work” items early.

Position rules correctly by class and/or RuleSet and actively use inheritance to prevent rule redundancy.

3. Do Nothing That is Hard

Use standard functionality as much as possible, especially in the initial project release. Avoid creating custom HTML screens or adding buttons and always enable auto-generated HTML on sections. Always use the standard rules, objects, and properties — reporting, urgency, work status, and other built-in behaviors rely on standard properties. Never add a property to control typical work or to manage the status or timing of work.

4. Limit Custom Java

Avoid Java steps in activities when standard PRPC rule types, library functions, or activity methods are available. Reserve your valuable time and Java skills for implementing things that do not already exist!

5. Build for Change®

Identify and define 10-100 specific rules for business users to own and maintain. Activities should not be on this list — use other rule types for business-maintained logic.

6. Design Intent-driven Process

Your application control structure must consist of flows and declarative rules. Use flow actions to prompt a user for input and present fewer than five connector flow actions for any individual assignment — if you need more than that, you need to redesign the process. To maximize reuse, create activity rules that implement only a single purpose, and call activities only as needed.

7. Create Easy-to-Read Flows

Flows should fit on one page and should not contain more than 15 SmartShapes (excluding Routers, Notify shapes & Connectors). If a flow has more than 15 SmartShapes, either create a subflow or use parallel flows to perform additional functions.

8. Monitor Performance Regularly

For best results, evaluate and tune application performance at least weekly. Use the Performance Analyzer (PAL) to check rule and activity efficiency — establish benchmarks early on; compare these results to follow on readings and correct application as required.

9. Calculate and Edit Declaratively, Not Procedurally

When appropriate, use declarative rules to calculate or validate property values. is calculated or validated, use declarative rules wherever appropriate.

We run a declare expressions rule instead of a Property-Set method in an activity, or a constraints rule instead of a validation rule.

10. Keep Security Object-Oriented Too

Your security design should be rule-based and role-driven, based on who should have access to each type of work. Use the standard access roles provided with PRPC only as a starting point and use RuleSets to segment related work for the purpose of introducing rule changes to the business, not as a security measure. Never code security controls in an activity.

Reviewing an Application

Objectives

At the end of this lesson, you will be able to:

- Use the Application Inventory landing page to review our application
- Identify problems to improve deployment

Things to Know

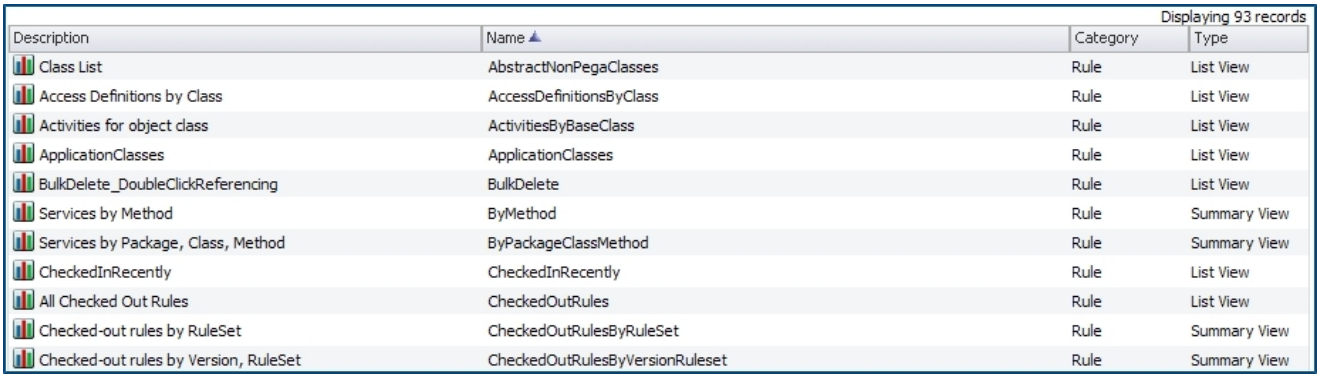
Once we've finished development of an application, it's recommended that we review the application for potential issues that may affect deployment. In this lesson, we'll look at some tools that help us review our applications.

The Application Inventory landing page contains three gadgets that help us to review the rules that make up our application.

- **Inventory Reports** — analyzes the rules that make up our application with various reports.
- **Six Rs** — categorizes application rules by functional category; also provides an application overview, including warnings and errors.
- **Heat Map** — analyzes the rules in our application graphically.

Inventory Reports

The Inventory Reports gadget lists PRPC reports that we can use to analyze the rules in our application. For example, we can review a list of the rules that are currently checked out, sorted by RuleSet version and name, or the number of processes for each work type.



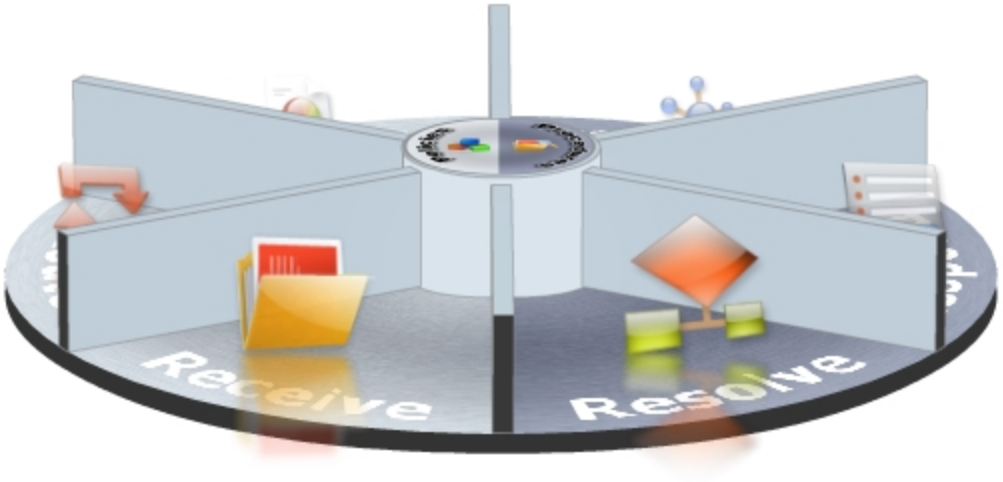
Displaying 93 records

Description	Name ▲	Category	Type
Class List	AbstractNonPegaClasses	Rule	List View
Access Definitions by Class	AccessDefinitionsByClass	Rule	List View
Activities for object class	ActivitiesByBaseClass	Rule	List View
ApplicationClasses	ApplicationClasses	Rule	List View
BulkDelete_DoubleClickReferencing	BulkDelete	Rule	List View
Services by Method	ByMethod	Rule	Summary View
Services by Package, Class, Method	ByPackageClassMethod	Rule	Summary View
CheckedInRecently	CheckedInRecently	Rule	List View
All Checked Out Rules	CheckedOutRules	Rule	List View
Checked-out rules by RuleSet	CheckedOutRulesByRuleSet	Rule	Summary View
Checked-out rules by Version, RuleSet	CheckedOutRulesByVersionRuleset	Rule	Summary View

Six Rs

The Six Rs gadget provides a summary of our application within the context of Pega's six functional capabilities.

Built on Application PegaRULES 06.02 , using 4 RuleSets, 256 Rules; *Last Updated 2/3/2012*
9 Users, 14 Classes , 42 Properties, 13 Specifications; 2 Critical, 2 Non-Critical Warnings

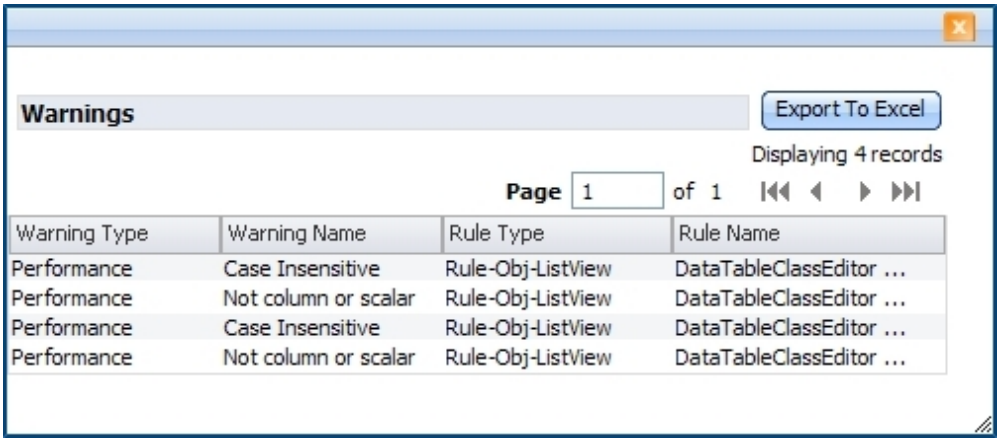


This gadget provides the following information about the current application:

- Built on Application, which may be a framework application or the PegaRULES application.
- Number of RuleSets used in the application (this number does not include Pega-prerequisites)
- Number of rules in the application
- Date of last update
- Number of users, classes, properties, specifications, critical and non-critical warnings

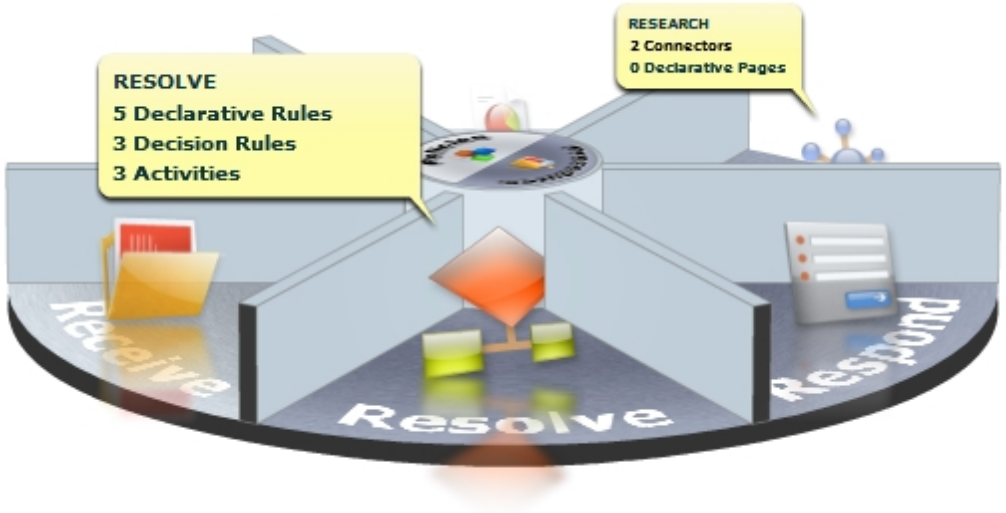
Built on Application PegaRULES 06.02 , using 4 RuleSets, 256 Rules; *Last Updated 2/3/2012*
9 Users, 14 Classes , 42 Properties, 13 Specifications; 2 Critical, 2 Non-Critical Warnings

We can click on each value in the header for more information. For example, clicking on the number of critical warnings displays a report listing the warnings and where in the application they occur as seen in the following example.



Warning Type	Warning Name	Rule Type	Rule Name
Performance	Case Insensitive	Rule-Obj-ListView	DataTableClassEditor ...
Performance	Not column or scalar	Rule-Obj-ListView	DataTableClassEditor ...
Performance	Case Insensitive	Rule-Obj-ListView	DataTableClassEditor ...
Performance	Not column or scalar	Rule-Obj-ListView	DataTableClassEditor ...

This gadget also provides a graphical breakdown of our application, displaying the rules in each "R" (functional capability). We can click on each capability to review the number of application rules for each capability. Clicking the rule types opens a report listing the respective rule instances.



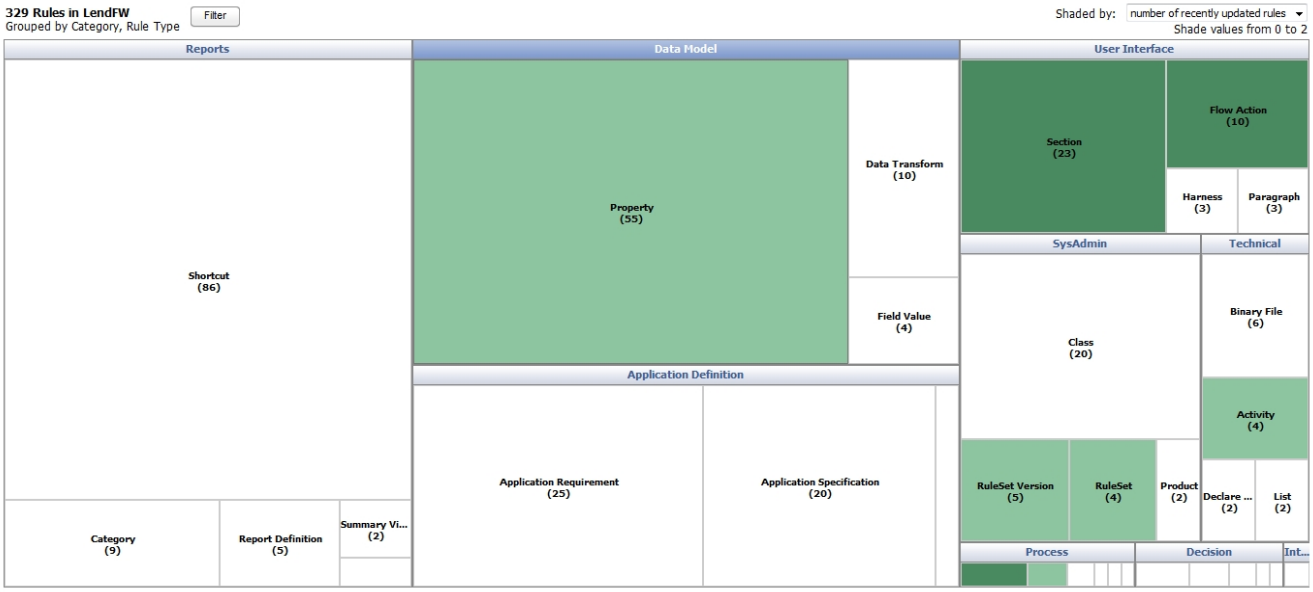
Heat Map

The Heat Map gadget shows the number of rules in each category, broken down by rule type, in a graphical format. The size of each box reflects the number of rules of a single rule type.

The heat map is shaded to graphically display one of the following counts, specified in the **Shaded by** drop-down list.

- The number of recently updated rules.
- The number of checked out rules.
- The number of warnings.

The darker the shade, the higher the number of filtered rules that are found in that rule type.



Click **Filter**, on the top left , to add or remove RuleSets and rule categories from the heat map.

We can:

- Left-click a rule type on the heat map to display a list view with all rules of the selected type in the application
- Right-click a rule type on the heat map to display a list view with all rules of the selected type in the application that match the shading selection for the heat map.

Documenting an Application

Objectives

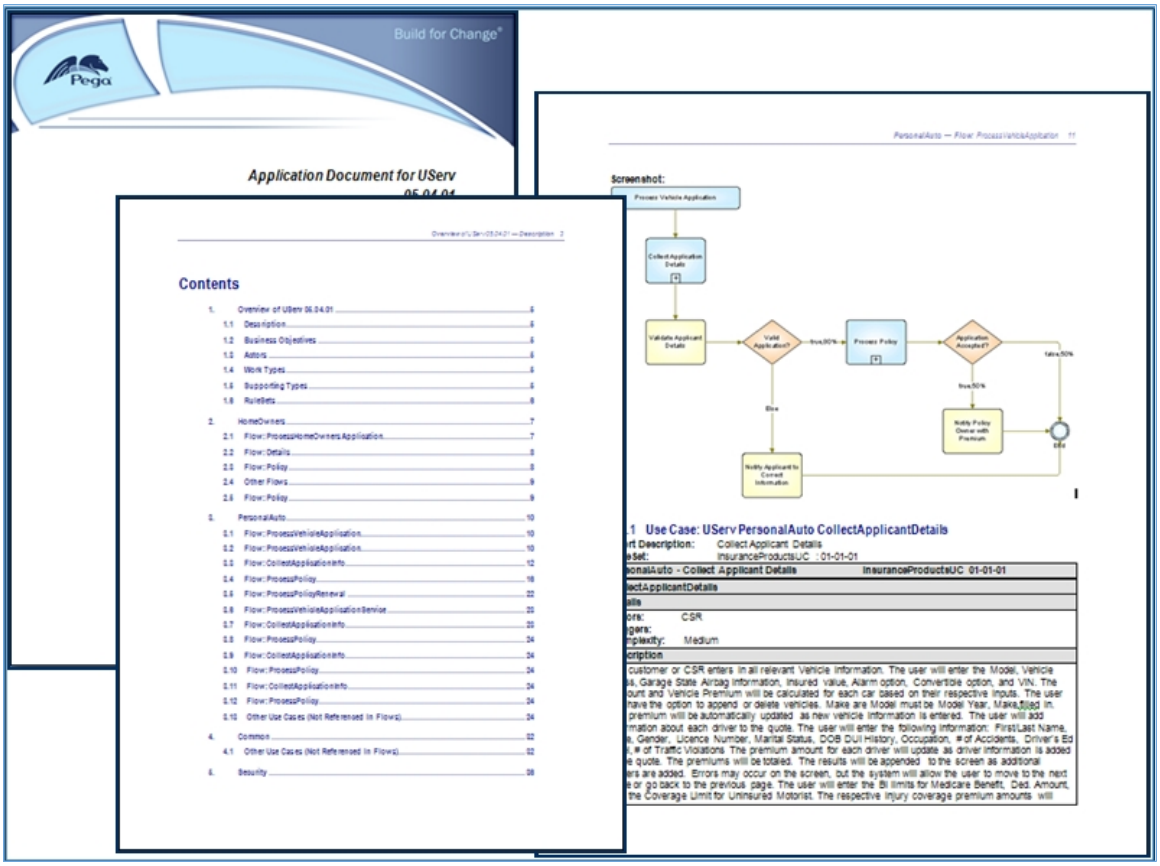
At the end of this lesson, you will be able to:

- Use the Application Document wizard to generate documentation for your application.

Things to Know

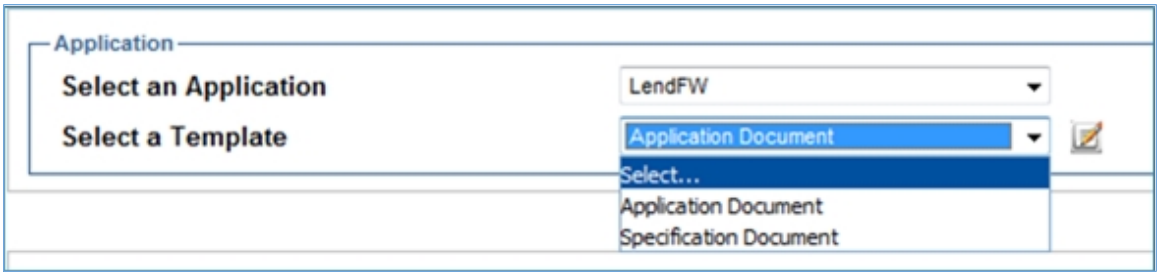
Generating Application Documentation

PRPC provides an automated documentation facility which assists business and system architects in providing detailed application documentation.



The Application Document wizard guides us through a process that includes selecting the aspects of the requirements or system to document. It creates a professional-looking Microsoft Word document that includes all the pertinent data that we choose to include, such as use cases, requirements, business objectives, actors, flow diagrams, UI screenshots and correspondence. The wizard uses information provided on rule forms, such as the short description and the contents of the rule itself, to generate application documentation. This structured document may support internal record-keeping responsibilities such as those required by Sarbanes-Oxley (SOX).

PRPC provides two fully-customizable templates, which allows us to create documentation tailored to the information we want to capture:



The Application Document wizard allows us to customize our documentation. For example, we can select the application to document and the template to use; select the RuleSets, work types, and criteria; order or suppress flows; define sample data and capture screenshots.

Once we have completed these steps, we can create our documentation. We can also skip these steps and create our documentation using only the default selections, if desired.